2005-03

Architectures for device aware network

Chung, Wai Kong

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ARCHITECTURES FOR DEVICE AWARE NETWORK

by

Peng Leong Seah
and
Wai Kong Chung

March 2005

Thesis Advisor:	Gurminder Singh
Thesis Co-Advisor:	Su Wen

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Architectures for Device Aware Network			**5. FUNDING NUMBERS**
6. AUTHOR(S) Peng Leong Seah and Wai Kong Chung			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

In today's heterogeneous computing environment, a wide variety of computing devices with varying capabilities need to access information in the network. Existing network is not able to differentiate the different device capabilities, and indiscriminatingly send information to the end-devices, without regard to the ability of the end-devices to use the information.

The goal of a device-aware network is to match the capability of the end-devices to the information delivered, thereby optimizing the network resource usage. In the battlefield, all resources – including time, network bandwidth and battery capacity – are very limited. A device-aware network avoids the waste that happens in current, device-ignorant networks. By eliminating unusable traffic, a device-aware network reduces the time the end-devices spend receiving extraneous information, and thus saves time and conserves battery-life.

In this thesis, we evaluated two potential DAN architectures, Proxy-based and Router-based approaches, based on the key requirements we identified. To demonstrate the viability of DAN, we built a prototype using a hybrid of the two architectures. The key elements of our prototype include a DAN browser, a DAN Lookup Server and DAN Processing Unit (DPU). We have demonstrated how our architecture can enhance the overall network utility by ensuring that only appropriate content is delivered to the end-devices.

14. SUBJECT TERMS Device Aware Network, Content Re-purposing, Heterogeneous Device, Capability Matching, Web Proxy, Web Browser, Device Profile.			**15. NUMBER OF PAGES** 101
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ARCHITECTURES FOR DEVICE AWARE NETWORK

Peng Leong Seah
Civilian, Ministry of Defense, Singapore
B. Eng. (Hons), National University of Singapore, 1996

Wai Kong Chung
Civilian, Ministry of Defense, Singapore
B. Eng. (Hons), Nanyang Technological University, Singapore, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2005

Authors: Peng Leong Seah

Wai Kong Chung

Approved by: Gurminder Singh
Thesis Advisor

Su Wen
Thesis Co-Advisor

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In today's heterogeneous computing environment, a wide variety of computing devices with varying capabilities need to access information in the network. Existing network is not able to differentiate the different device capabilities, and indiscriminatingly send information to the end-devices, without regard to the ability of the end-devices to use the information.

The goal of a device-aware network is to match the capability of the end-devices to the information delivered, thereby optimizing the network resource usage. In the battlefield, all resources – including time, network bandwidth and battery capacity – are very limited. A device-aware network avoids the waste that happens in current, device-ignorant networks. By eliminating unusable traffic, a device-aware network reduces the time the end-devices spend receiving extraneous information, and thus saves time and conserves battery-life.

In this thesis, we evaluated two potential DAN architectures, Proxy-based and Router-based approaches, based on the key requirements we identified. To demonstrate the viability of DAN, we built a prototype using a hybrid of the two architectures. The key elements of our prototype include a DAN browser, a DAN Lookup Server and DAN Processing Unit (DPU). We have demonstrated how our architecture can enhance the overall network utility by ensuring that only appropriate content is delivered to the end devices.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

We would like to express our heartfelt thanks to the following people for their kind support and assistance that leads to the successful completion of this joint thesis.

Many thanks to Prof. Gurminder Singh and Prof. Su Wen for their helpful advice and directions. Their guidance and encouragement kept us on track throughout our endeavor.

We would also like to thank Mr. Arijit Das and Mr. John Gibson for providing the necessary resources and valuable inputs during our numerous group discussions.

Finally, to our families and friends for your never-ending support and patience.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

In today's heterogeneous computing environment, there exists a wide variety of computing devices with varying capabilities that need to access information in the network. These devices (e.g., Personal Digital Assistant (PDA), desktop computer, notebook computer, cell phone and a host of networked embedded systems) may have extremely differing capabilities and resources to retrieve and display the information from the network. In addition, such devices may be connected to the network through various means, each with differing capacity and bandwidth. As a result, one form of information apt for a particular class of device in a particular network environment may not be optimal for other classes of device in another networking environment.

Unfortunately, the network we have today is not designed to address such disparities that exist among the end devices. The network is like dumb pipes, one which simply route the data packets to the requesting devices, without regard for the device's ability to use such data packets. As a result, end device may end up wasting resources retrieving information it cannot use from the inert network. This is especially critical for constrained devices (e.g., PDA, cell phone) which have relatively limited resources. For example, in the case of a PDA retrieving a large image from a server, the resources required to process the large amount of data received from the server may exceed the capability of the PDA. This may result in the PDA discarding the received data or the PDA committing too much of its resources to display the image, causing it to "crash". In both cases, resources such as processor time cycle, memory and battery power of the PDA are wasted to process unusable data. Moreover, the network also spends a significant portion of its resource to transport and process unnecessary traffic [13].

The root of the resource wastage issue is due to the current network not able to distinguish contents that are not suitable for the end devices. A possible solution is to enhance the network capability to make it aware of the end devices capabilities. Known as a Device Aware Network (DAN), DAN matches the content of the data with the capability of the destination end device. If the data is deemed unusable, DAN will

prevent the data from entering the network or transform it to a format that the end device can process. Therefore a DAN avoids the inefficiencies that consume unnecessary end device and network resources. Although a DAN works well in a wired network, its strength lies in providing an efficient environment for wireless and mobile applications where the resources – both the network and end devices – are limited [13].

B. PURPOSE OF STUDY

In this thesis, we study the use of a DAN framework to enhance the efficiency of a network so that it is able to distinguish the capabilities of end devices. By doing so, it is able to deliver the appropriate information to the end device and prevent wastage of resources.

The first phase of the study involves performing background study. During this phase, we will research and study those technologies that are relevant to the DAN framework. We intend to harness the strengths of various approaches and technologies to develop the DAN framework. We will also conduct a requirements study for the DAN framework to identify the broad functionalities required.

The next phase of the thesis involves drafting architecture designs for the DAN framework. The artifacts from the background study should provide us with sufficient knowledge and familiarity to propose various architecture approaches. We will also perform requirements analysis on the proposed designs.

Finally, the last phase requires us to develop a prototype of the DAN framework to demonstrate its conceptual viability. We shall harness the strengths of the various designs, and create the blueprint upon which we build the prototype. This will be followed by the development of a skeletal prototype of the DAN framework. The thesis culminates with a prototype demonstration to verify the concept of the DAN framework.

C. ORGANISATION OF THESIS

The organization of the thesis follows closely to the various phases of work identified in the previous section.

Chapter II discusses the various key technologies that are relevant to the DAN framework. It will also document the requirements artifacts that highlight the key functionalities of the DAN framework, as well as defining the scope of the DAN framework.

Chapter III proposes the alternative designs for the DAN framework. It first identifies the various design considerations upon which the designs were derived from. Subsequently, it describes each design, and discusses their salient features. A critical analysis of each design against the requirements is also conducted.

Chapter IV documents the prototype implementation. The chapter begins with an analysis of the challenges involved in designing and building the prototype. It discusses the DAN framework design as well as the implementation. Finally, it documents the result of the prototype demonstration.

Chapter V summaries the issues and lessons learnt we gathered throughout the thesis work, especially the prototype implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND STUDY

A. CHAPTER OVERVIEW

This chapter begins with a discussion on technologies that are relevant to the development of a DAN framework, and then continues to list the functional requirements of DAN framework, which is compiled as a result of the requirement study.

B. RELEVANT TECHNOLOGIES AND IMPLEMENTATIONS

This section briefly describes those technologies that are relevant to DAN.

1. JINI Technology

Jini is a distributed computing technology that builds upon the existing Java application environment. It extends the Java application environment from a platform-centric to a network-centric environment. The Java application environment is a good candidate for distributed computing due to its ability to move both code and data from machine to machine. It also capitalized on Java's security and strong typing features that enables Java classes to be run, with confidence, on a virtual machine even when a class does not reside physically on that machine. The result is a networked system that allows objects to transit from machines to machines and codes be executed in any part of a Jini network [1].

The central theme of Jini is the notion of a service. Members of a Jini system, consisting of hardware devices or software components, which provide some functionality, can be deemed as a service. In a Jini system, different services may federate to perform a certain task. Examples of Jini services include, among all things, devices like printer, fax machines or hard disk for storage; software such as applications or utilities; information from database server or data files [1]. Programmatically, a service is declared as a Java Language interface that defines operations that can be provided by the service. In most cases, it is also be identified by this interface [2]. Based on this interface, different vendors or service providers can implement the same services using different Java implementation.

A Jini system, also known as a federation, is made up of clients (service consumers) and services (service providers), all communicating using the Jini protocol (based on the Java Remote Invocation mechanism) [2]. For a service to be used by a client, it must first register itself with a lookup service. The lookup service serves as a central repository for all the available services within a federation. The registration of service can be accomplished in the following ways. If the service provider knows the location of the lookup service, it can send a unicast message to the lookup server and register itself with the server. If the location is not known by the service provider, it can send a UDP multicast to discover the lookup server. If there is a lookup server listening to the request, it will response to the service provider request and the registration can take place [2]. This is, in Jini terms, known as the discovery process.

Once the location of the lookup service is discovered, the service is ready to join the federation. To begin the join process, a service object for the service is created and loaded onto the lookup service. The service object consists of the Java Language interface that defines the methods a client can invoke as well as other attributes that describes the services [1]. From this point onwards, any client who wishes to connect to the service can locate it using the discovery mechanism discussed above. The lookup service, in turn, will return a copy of the service object to the client via a network when it receives the lookup request. The service object will run in the Java runtime environment of the client and acts as a proxy for the requested service. All interactions between the client and the service are made via the service object.

The discovery mechanism in Jini is an example to provide location transparency for servers or services. In a Device Aware Network (DAN), we could adapt this concept to make the location of DAN related resources variable. Location related information, such as IP address of DAN resources can be stored in a lookup server. A client can perform a lookup for the location of the DAN resources before accessing these resources.

2. Content Repurposing

With the proliferation of wireless mobile devices and advances in wireless communications, web contents are widely accessed by a large variety of devices, other than the traditional desktop and notebook computers. Devices like Personal Digital Assistant (PDA), cell phones, etc, have become a convenience means for mobile users to access the World Wide Web. However, due to the limited display real estate of these mobile devices, most contents designed for the web are not optimized for display on these low resolution screens.

Content re-purposing is one of the possible techniques to tackle with the issue of providing the right type of data to the right device. The central idea in content re-purposing is to maintain a single copy of the content and automatically perform in real time to repurpose the content to the required format that is optimized for a certain category of client devices [3]. Several approaches to content re-purposing have been proposed and implemented, namely the client-based, server-based and proxy-based approach.

In the client-based approach, the content is delivered to the client even though the client is incapable of displaying the content. Some pre-processing is required to repurpose the content so that it can be displayed meaningfully by the client. The advantage of this approach is that the server and the existing infrastructure do not need to be modified when new devices are introduced. However, many devices are limited in the computational power. In most case, the content may not be able to be displayed by the client, even though an attempt to repurpose by the client is made. For example, high resolution video may overwhelm a device powered by a slow processor. Moreover, the network bandwidth is wasted to transfer unusable data.

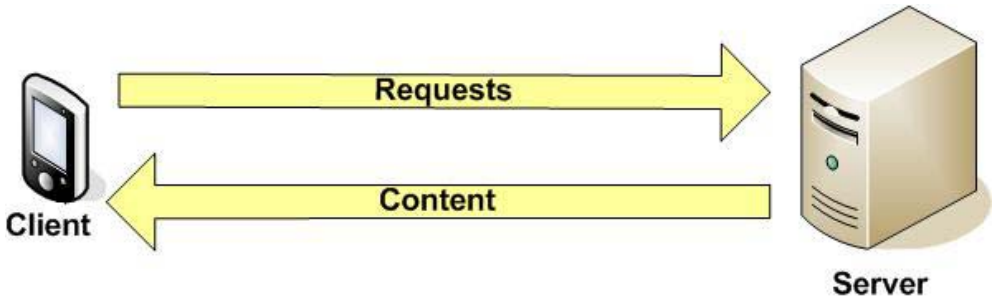

Figure 1. Client-based Content Repurposing (After Ref. [13])

The server-based approach involves the source of the content to perform the necessary repurposing tasks. The server, being more resourceful, can transform the data format with ease. It can also decide that if no available format is suitable for the requesting client, it can stop sending the content altogether. This will prevent unnecessary network traffic as we have seen in the client approach. However, the drawback is that this will involve modifications on servers to achieve this.

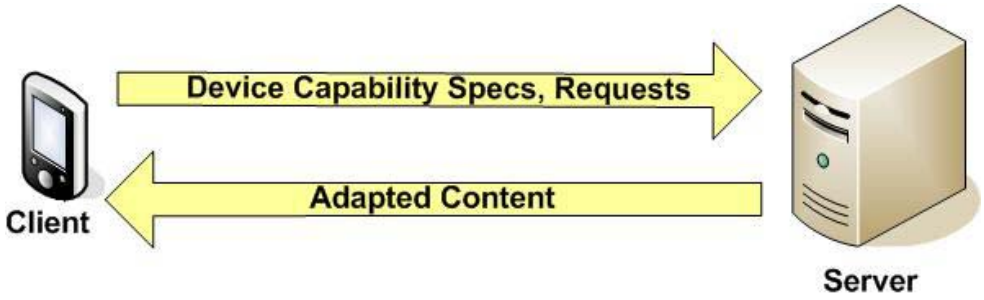

Figure 2. Server-based Content Repurposing (After Ref. [13])

The proxy-based approach is deemed to be in the middle ground between the client- and server-based approaches. It can off load the repurposing tasks from the server and hence does not require any changes to the server's configuration. Similar to the server-based approach, the proxy can also stop any unusable content by the client from becoming part of the network traffic. On the other hand, the proxy-based approach requires an additional hop before the content is delivered to the client. This will incur additional delay for the total time taken to deliver the content.

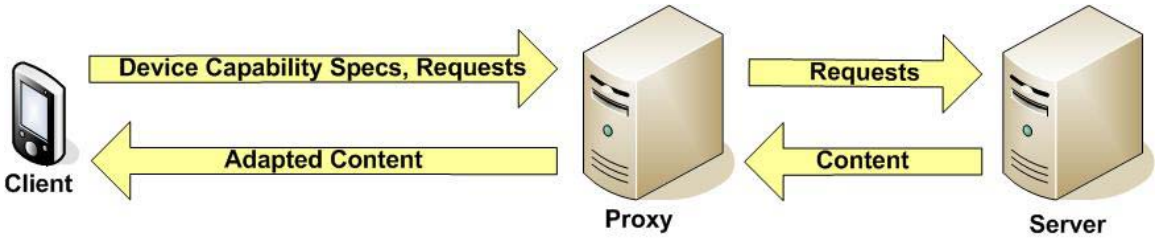

Figure 3. Server-based Content Repurposing (After Ref. [13])

For the case of a DAN, the proxy-based approach is preferred as it does not require existing servers to make any changes. Moreover, using a proxy will give the flexibility to develop a new protocol to support DAN without worrying about the DAN protocol co-existing with other protocols.

C. REQUIREMENTS STUDY

We conduct a requirements study to gather the functional requirements of the DAN framework. The objective of this study is to help us better understand what is required of the DAN framework, and it shall serve as a reference for the subsequent phases of thesis work. We felt that this is necessary as DAN is a relatively new concept, and it would help the team to align our works. We have chosen Use-Case methodology for this study.

As our purpose is not to gather comprehensive requirements typical in software development cycle, we will only attempt to capture the relatively broad requirements. Specifically, we will only perform up to use case model survey, which will capture the broad overview of the use cases (i.e., key functional requirements), actors and the framework's scope.

1. User Case Model Survey

a. Actors

The actors represent external entities that will interact with the DAN framework. They have been identified to ensure that the requirements gathered during this phase are as complete and accurate as possible.

Actor	Description
Client	Refers to any user who use computing devices (e.g., desktop computer, notebook, PDA, cell phone) to retrieve information from information servers through the network.
Information Server	Refers to any devices that are able to supply the information required by client terminals through the network.
Network	Refers to the infrastructure that interconnects client terminals and information servers for the purpose of data exchange. That is, its enable the client terminals to request for information from information servers and for the latter to response with the requested information back to the

Actor	Description
	requesting client terminal. The network may be wired or wireless, internet or intranet.
Service Provider	Refers to a node in the network that is capable of providing relevant services to complement the DAN framework, such as repurpose specialized content, and perform capability matching. The service provider may either perform the tasks directly, or provide the necessary software artifacts to the DAN framework for processing.
DAN Processing Unit (DPU)	Refers to the intermediary nodes, which reside on the network, to facilitate information exchange between client terminals and information servers in an efficient manner.

Table 1. List of Actors

b. Overview of Use Cases

The following uses cases were identified for DAN. These use cases, which represent the essential DAN functions, serve as a functional abstraction for the subsequent design and prototype implementation phases.

Use Case	Description
Register Device	The use case begins when a client register a new device to the DAN framework to upload the device's profile and user preference to the framework prior to initiating a session to the information server. This information will be used by the framework to perform capability matching. The device profile uploaded may be dynamic or static. Unlike dynamic profile (e.g., battery level, network media), static device profile (e.g., device class, display resolution and multi-media capabilities) does not change over time. User preference may also be uploaded to provide personalized level of content

Use Case	Description
	repurposing.
Register Service	The use case begins when a service provider registers the service it is able to provide to the DAN framework. Such supplementary services include request forwarding, content repurposing and capability matching.
Join Network	The use case begins when a client device joins the DAN framework. Both the DAN framework and the device shall be appropriately initialized to prepare the DAN framework to service the device in the current environment. This initialization is performed prior to the client device initiating a session to the information server.
Lookup Service	The use case begins when a client device needs to look up for an appropriate component of the DAN framework. The lookup may be explicit or implicit. In the former, the components may be located through explicit exchange of messages with the DAN framework (i.e., Lookup Server). In the latter, a built-in mechanism helps to locate the component without any additional message exchange (i.e., components are located when the client sends request for information to the information server).
Repurpose Content	The use case begins after the DAN framework determines that there the client device is not able to use the content returned by the information server, or that the user only requires a subset of the content. It will attempt to repurpose the content according to specific business rules and user preferences so as to reconcile the differences. This may be through some published services or generic rules.
Survey Device/Network	The use case begins prior to content repurposing. The DAN

Use Case	Description
Status	framework will survey the network, as well as probing the client device dynamic profile to determine if they can support the transportation and utilization of the information to be delivered.
Initiate Session	The use case begins when a client device tries to connect to a information server. The DAN framework may attempt to negotiate a compatible protocol to facilitate the content retrieval. This is especially useful for cases when the client device only supports very limited protocols (e.g., embedded system, which does not support the HTTP standard), tries to get information from internet).
Negotiate Capability (or Capability Matching)	The use case begins before the DAN framework forwards the requested information from the information server to the requesting client device. The DAN framework will determine if the content matches both the capability of the client device, and the user's preferences.
Locate Mobile Device	The use case begins when DAN framework needs to locate a specified client device to deliver the request content (e.g., deliver subscribed periodical message).

Table 2. List of Use Cases

c. Use Case Diagram

The diagram below depicts the interaction between the actors and the use cases. The rectangle depicts the scope of DAN framework, while the individual eggs represent the key functions (i.e., use case). The actors are represented outside the DAN framework as they are external to the development of the DAN framework. The interaction between the actors and uses cases are represented by the lines that connect them.

For the purpose of this thesis work, we will not be proceeding to the detailed Use Case specifications. The following design and implementation phases will be guided by the broad requirements specified in the Use Case Model Survey. This is to allow us to proceed rapidly to the implementation phase to evaluate the alternative designs, and provide a proof of concept of the DAN framework.

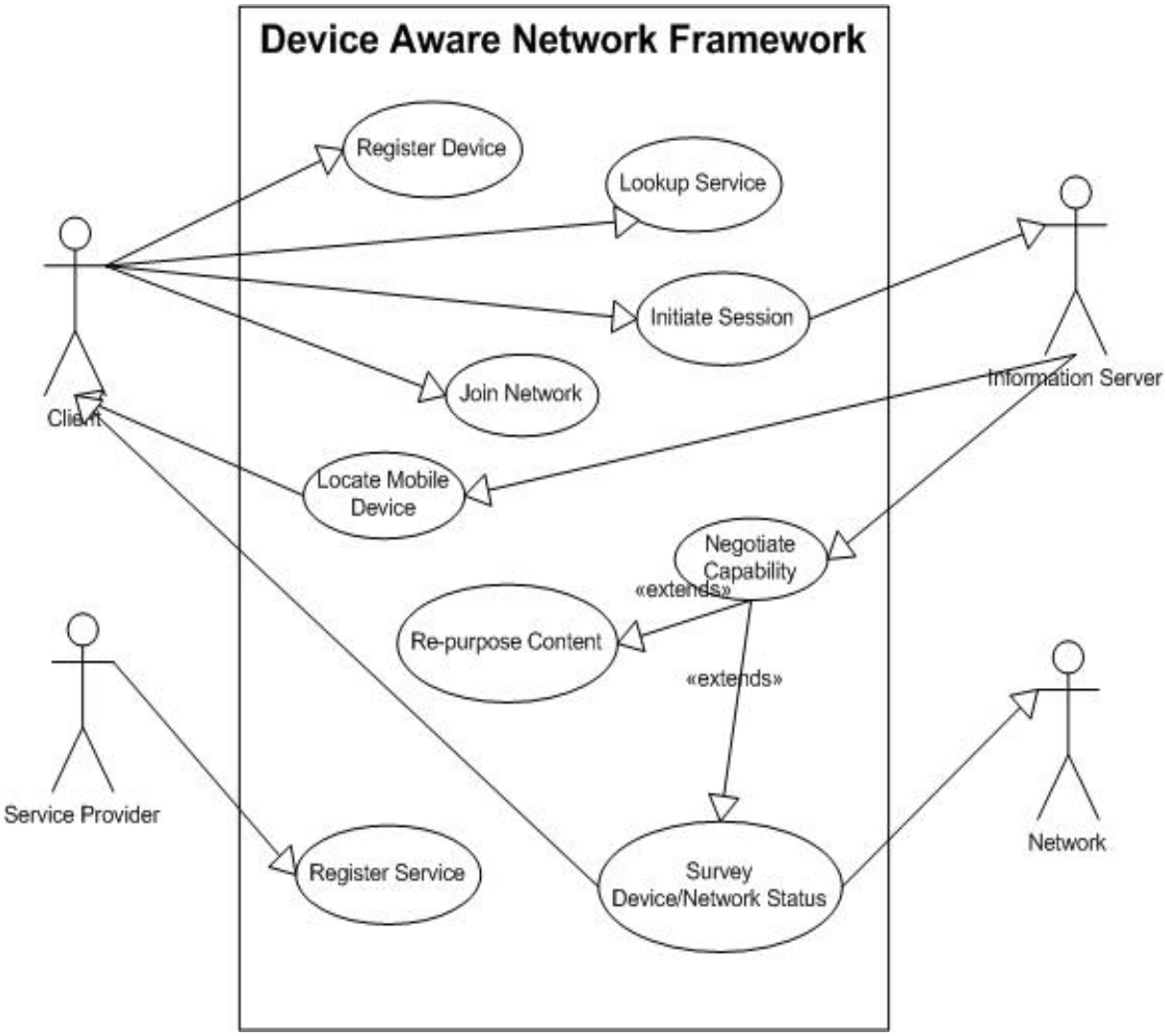

Figure 4. Use Case Diagram for Device Aware Network Framework

THIS PAGE INTENTIONALLY LEFT BLANK

III. ARCHITECTURES FOR DEVICE-AWARE NETWORK

A. CHAPTER OVERVIEW

This chapter discusses the architectures for a DAN framework. It first presents the key design considerations in order to fulfill the DAN requirements. This is followed by the study of two possible DAN architectures, namely the Proxy-based and the Router-based approaches.

B. KEY DESIGN CONSIDERATIONS

Several considerations went into the design of DAN framework. These considerations were selected to ensure that the DAN framework architecture can achieve the objectives in the most efficient manner, and are practical for widespread adaptation. Though we started out with a large list of possible designs, we selected two for implementation and testing.

The following sections list our key considerations for the DAN architecture.

1. No Additional Hardware

The DAN framework should not require deployment of additional hardware. At the client end, the user should be able continue to use his existing computing device (e.g., laptop, PDA, cell phone) when operating in a DAN environment, just like in a non-DAN environment. He shall not need to add any peripherals nor upgrade his existing device. Also, DAN should not mandate the use of any additional hardware in the network for the sole purpose of accommodating the DAN processing.

Likewise, the content or service provider shall not need to install additional servers or modify existing setup to comply with the DAN framework.

2. Minimal Changes to Server and Client Software

Software modules may be required to be installed in existing hardware to effect the DAN environment. Such software modules shall be compatible with the other software applications already present in the hardware. In particular, it should not require

the client to switch to another browser nor use a different network protocol. The client should be able to continue using the suite of software applications as before.

However, no software changes shall be required on the content or service provider. The rationale for this is purely practical - it would be too expensive to expect all the content providers to make any changes to their systems.

3. Little or No Human Intervention Required

In order to entice users to operate in the DAN environment and enjoy the resulting benefits, the client interface should require as minimal a user intervention as possible. Beside the occasional update to preferences or to override the default DAN settings; the end user should not have to perform any additional tasks to operate in a DAN environment. For example, when using a browser to surf the web, the user will continue to perform the usual tasks to navigate to the desired URL, while the DAN modules will work in the background to ensure that the content received is in tandem with the user's preferences and device's capabilities.

4. Maximize Efficiency for Constrained Devices

The design should strive to maximize the efficiency of those constrained area (e.g., mobile devices, wireless network), while ensuring that the overall efficiency of the network.

One way of improving efficiency is to transfer tasks from resource constrained devices to more capable devices. This way, the efficiency is improved since the more capable devices will take less resources (e.g., time) to complete the tasks.

The other way to improve efficiency is to optimize network traffic.

5. Accommodation for Heterogeneous Devices

One of the objectives of the DAN is to address the mismatch between the content delivered and the receiving client terminal. This mismatch is caused, in part, by the increasing diversification of computing devices in the market, and compounded by the indiscriminating behavior of the network, which is only concerned with routing the traffic

efficiently through the most optimal path and is indifferent to whether the data is usable by the receiving terminal.

Since DAN is designed to address the problems posed by heterogeneous devices, it is imperative that the client solution is designed to be portable across multiple platforms. This consideration not only applies to commercially available computing devices (e.g., PDA, notebook, cell phone), but also to military embedded system such as weapon system. To achieve this, it is important to find the common ground among such wide disparities of devices.

C. PROXY-BASED APPROACH

This section describes the Proxy-Based approach, as one of the two alternative designs for the DAN framework architecture.

1. Overview of Proxy-Based Approach

The proposed proxy-based approach is modeled after the web architecture that is commonly used today. In this approach, DAN will make use of a proxy-like agent, called DAN Processing Unit or DPU, which is analogous to the web proxy that resides in the local area network and operates at the application layer. It will be deployed to serve all the clients within the intranet. The DPU needs not be a dedicated machine, and can co-locate with existing machines on the network. All the client terminals will be registered with the appropriate DPU within the intranet. The client's static device profile will also be updated to the DPU during the registration.

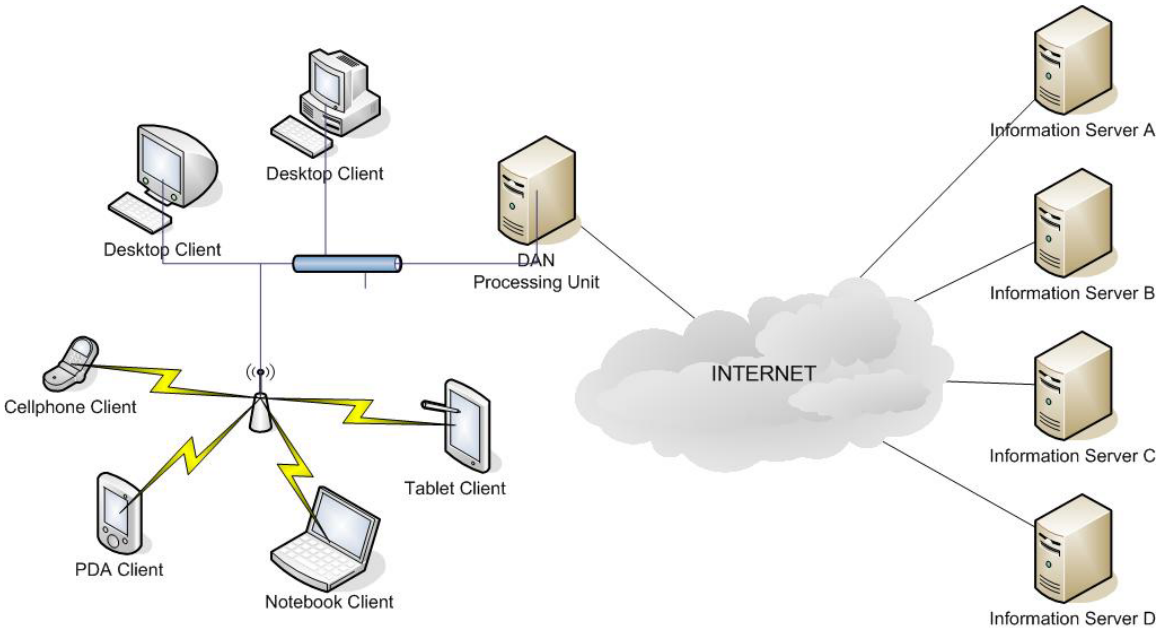

Figure 5. Schematic Diagram of Proxy-based Design

Instead of connecting to content providers (both internal and external) directly, client terminals will forward the information request to the designated DPU. The DPU, in turn, submits the information request to the information provider on the client's behalf. This step ensures that the DPU receives the requested content directly from the information provider, instead of the requesting client. Alternatively, the client may submit the request to the information provider directly, but directs the result to be returned to the DPU. However, such redirection may not be compatible with most standards, such as the HTTP specifications. The dynamic device profile may be transmitted to the DPU during this step.

Upon receipt of the information, the DPU then determines if the content is appropriate for the requesting client, based on the client's device profile (both static and dynamic) and user's preferences. It may need to repurpose the content in order to match the content to the capability of the receiving client. Finally, it passes on the resultant customized content to the client.

2. Salient Features of Proxy-Based Approach

A salient feature of this approach is it operates predominantly at the application layer of the Open System Interconnection (OSI) stack. This ensures that the solution will not modify the underlying protocols such as TCP/IP, and ensure the solution is compatible with existing infrastructures that comply with the OSI standards. In addition, it is unaffected by changes in the underlying protocols, such as the impending upgrade from IPv4 to IPv6.

However, one drawback of working at the highest level of the OSI stack is the trade off in efficiency. The DAN framework has to communicate with the client device through message exchange. Hence, it will incur higher overheads as compared to embedding such messages in the network protocols of the underlying layers.

Another salient feature is its close resemble to the web model. The use of proxy-like intermediate node within its intranet means that the design will be compatible with the security solutions employed in most networks.

3. Design Analysis of Proxy-Based Approach

a. Device Registration

The client device only needs to upload selected profile information to the DAN framework prior to using it to retrieving information from the information server. The assumption is that the device class is already pre-registered in the framework so that most of the fixed and/or common profile information is already available within the framework.

The client device can upload the profile information via any of the several standards such as User Agent Profile Specification (UAProf) [5], Resource Description Framework (RDF) [6] or by using open formats such as XML. Alternatively, this information can also be embedded in the protocol used by the request for information (e.g., embed this information in the HTTP protocol for web request).

b. Service Registration

Every service provider needs to register the service it is able to provide with the DAN framework. This is important so that the DAN framework is able to provide the client devices with an appropriate list of services available in the network and the corresponding service provider. Client device needs this information when submitting information request, so that the DPU knows exactly where to obtain such services for capability negotiation and content re-purposing

c. Joining the Network

The initialization process involves probing for the DPU within the intranet. This process is similar to the Dynamic Host Configuration Protocol (DHCP) mechanism, where the client device will broadcast the search message until a DPU responds with the required configuration information.

Upon receipt of the configuration information, the device will automatically configure itself so to join the DAN framework. In this approach, the DPU is assigned to be the dedicated proxy for the client device. Hence, this device setting will remain valid until the client device leaves its existing networking environment (e.g., subnet).

The current DPU may also update the home DPU or a central registry on the current location of the newly joined client device. This is to assist the framework in locating the client device when the need arises.

d. Lookup Service

The functionality is subsumed under the "Joining Network" function.

e. Repurpose Content

The dedicated DPU will repurpose the content for the client device it is assigned to. This assignment takes place when the client device joins the network. The DPU may repurpose the content using the appropriate policies in its cache, or it may download the policies it requires from service providers in the network.

f. Survey Device/Network Status

The dedicated DPU will probe the client device for dynamic device profile whenever necessary. This is done through message exchange with the client device. The network probing is conducted in similar manner.

g. Session Initiation

When the client device initiates a request for information, this request will be forwarded to the dedicated DPU. The DPU, in turn, re-route this request to the appropriate information server. Thereafter, the DPU will serve as the client's proxy, and liaise directly with the information server.

h. Capability Negotiation

The dedicated DPU performs this task in a similar manner to re-purposing content.

D. ROUTER-BASED APPROACH

1. Overview of Router-Based Approach

This approach makes use of network routers to implement the DAN framework. The gist of the idea is to leverage on the existing edge routers who are closest to the intended information servers. Besides being able to reuse the existing infrastructure, having the DAN functionality being administered close to the information server can yield considerable improvement in network efficiency.

Unlike the proxy-based approach, the client device can send a request for information directly to the intended information server. This network packet is specially marked so that the information server's edge router can pick up this request. From the additional information embedded in the packet, the edge router is able to extract the information pertaining or leading to the detailed device profile of the requesting client. It will contact the appropriate service providers to download the proxy codes required to

perform capability negotiation as well as content re-purposing for this specified client device. In the meantime, the request is sent through to the specified information server.

When the information server returns the requested information, the data packets are automatically routed to its edge router. The edge router will detect and extract the relevant packets meant for the requesting client. These packets will be assembled to be compared against the requesting client's device profile, and shall be re-purposed if any mismatch is found. Finally, the edge router will encapsulate the resultant information in the appropriate protocol, and route them directly to the requesting client.

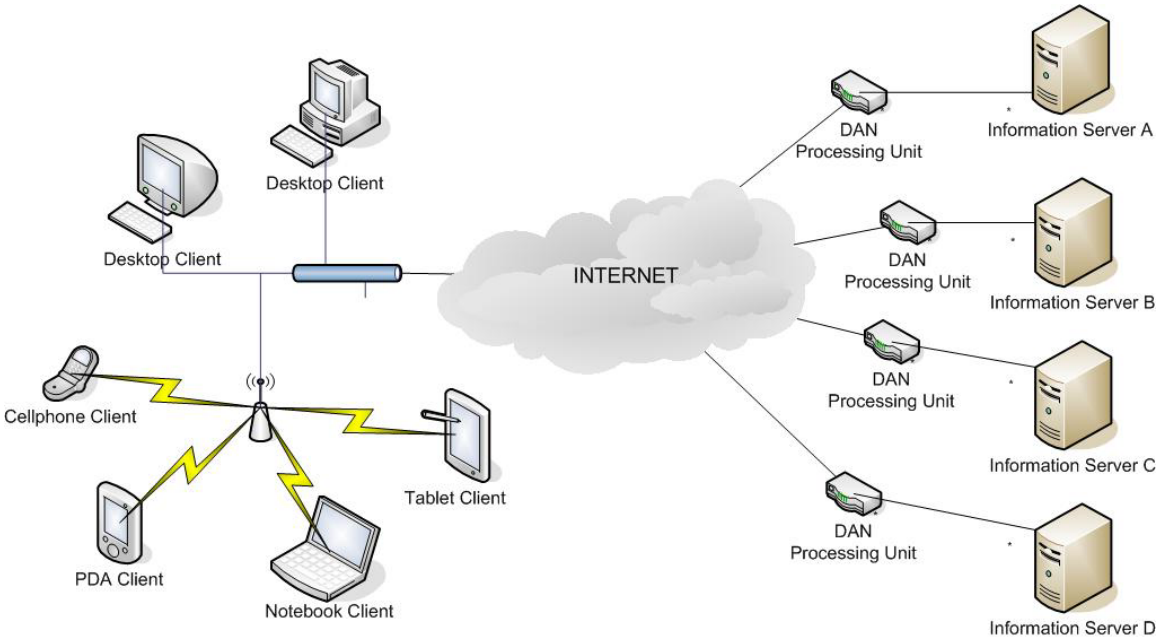

Figure 6. Schematic Diagram of Router-based Design

2. Salient Features of Router-Based Approach

Unlike the proxy-based approach which operates at the highest layer (i.e. application layer), this approach targets the lower layers of the OSI stack (e.g., network and transport layer). Instead of message exchange, the DAN framework can communicate with client devices by embedding such messages in existing network packets. This greatly improves the efficiency of the framework.

A drawback of this approach is that it requires modification to the network protocol. The process of rectifying existing protocols and pushing for its widespread

adaptation can be very tedious and time-consuming. The slow adaptation of IPv6, given its clear benefits, amply illustrates the difficulties involved.

Another salient feature is the DPU may be different for different information requests. Since the edge router and information server form a paired relationship, information requests going to different information servers will have to be serviced by different edge routers. The operational implication is the framework has to re-search for the correct DPU each time the client device submits an information request. Theoretically, this task is inexpensive since the search mechanism is built into the network protocol. This cannot be said for the other tasks to download proxy codes for capability negotiation and content re-purposing.

3. Design Analysis of Router-Based Approach

This section analyses the design against the broad functional requirements.

a. Device Registration

Like the proxy-based approach, it is assumed that the common profiles of the class of the client device already exist in the framework. Thus the client device only need to upload dynamic profile information and user's preferences whenever appropriate (i.e., submit an information request). Such information may only need to be uploaded during submission of information request, and is embedded in the request's network packets.

b. Service Registration

Every service provider needs to register the service it is able to provide with the DAN framework. This is important so that the DAN framework is able to provide the client devices with an appropriate list of services available in the network and the corresponding service provider. Client device needs this information when submitting information request, so that the DPU knows exactly where to obtain such services for capability negotiation and content re-purposing.

c. Joining the Network

The initialization process requires the client device to query the DAN framework for providers of specific services (e.g., providing proxy codes for capability negotiation). This information is required by the client device when submitting information request.

There is no need to configure the DPU information in the client device, since such information is dynamic, depending on which information server is responding to the information request.

d. Lookup Service

The DAN framework provides a generic lookup service to address the client device's query for proxy services available in the framework.

e. Repurpose Content

The edge router needs to assemble the network packets returned by the information server to reproduce the content prior to content repurposing, if necessary. It may obtain the proxy codes required for this task from the designated service provider, as specified in the client's information request. After the content is re-purposed, it will send the modified content to the requesting client.

f. Session Initiation

The client device initiates a session with the information server, via the edge router designated as the DPU, when it submits the first information request. It will embed the proxy codes information, required by the DPU, in the first information request of each session. This information is necessary for DPU to download the proxy codes necessary to perform capability negotiation and content re-purposing

During the session, the client device will not need to re-send any of the administrative information since they will be cached by the edge router. To avoid over burdening the router's memory, time-out will be set for such caching. When time-out is

reached (i.e., the client device did not submit any request during the period equivalent to the time-out), the edge router may purge such information from its cache.

g. Capability Negotiation

Like content repurposing, the edge router needs to re-assemble the network packets returned by the information server. It will perform the capability negotiation using the proxy codes downloaded from the specified service provider.

E. SUMMARY

So far, we have discussed the key considerations for the architecture of the DAN framework. Based on these considerations, we developed two possible approaches (i.e., Proxy-based and Router-based) for the DAN framework. The two fundamental differences between these approaches is the location of the DPU and the stack layer at which the DAN functionality operates.

For the Proxy-based approach, the DPU is always located in the proximity of the client device. It provides DAN functionality to the client device through application layer services. This design is similar to the web proxy model used in many networks to provide a centralized channel to the internet.

On the other hand, Router-based approach places the DPU closer to the information server (e.g., web server), and operates more dominantly in the network layers. The concept of this design is more radical than the proxy-based one. It requires expansion of the network protocols as it targets at enhancing the network's inherent capability.

In the next chapter, we shall present our prototype design. The design is based on a hybrid of both approaches we discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. PROTOTYPE DEVELOPMENT

A. CHAPTER OVERVIEW

This chapter provides the insights to the design and development of the prototype that is used to show the feasibility of the DAN framework. It continues with the documentation of the flow of events that take place during the prototype demonstration.

B. PROTOTYPE OVERVIEW

1. Objective of Developing Prototype

The objective for developing the prototype is to demonstrate the viability of the DAN framework. We intend to verify, through the prototype, that the proposed DAN framework can enhance the network with the capabilities to match the contents and capabilities of client devices in order to optimize resources of constrained devices and network traffic.

During the design phase, we also encountered several technical issues that could potentially impede the implementation of the DAN framework. This development work will provide us an opportunity to better study these issues, and to try out various solutions.

To facilitate the prototype development, we narrowed the scope of the DAN framework to support web technology. The DAN framework is intended to support a wide range of information exchange technologies or specifications. However, due to the limited resources, it is not feasible for us to try to take on all the technologies at one time. We chose the web technology since it is the predominant platform for information exchange.

Lastly, it should be noted that the prototype does not amount to having a fully qualified DAN framework. This prototype is only the beginning and a small step towards realizing an operational DAN framework. The prototype will allow us to examine the potential operational issues, explore alternative designs and solutions, and more importantly, provide the test bed for future work.

2. Challenges for Client Device

The key challenge for the client device is to discover the right DPU, and self configures the browser to connect to this DPU. Other challenges include updating the DPU on its static and dynamic profile.

For the proxy-based approach, we can follow the same mechanism that internet browser uses to search for proxy server dynamically. However, this mechanism cannot be used for the router-based approach. The client device cannot use broadcast mechanism for this purpose since the DPU-capable router, e.g., the server's edge router, is usually not in the same broadcast domain.

In addition, the designated DPU changes for different information server. This means that the DPU can only be determined when the user specifies the information server (i.e. web site via Universal Resource Locator or URL). This requires a very different setup from the regular browser, which normally has a fixed proxy setting independent of the websites it navigates to.

The other challenge is the need to have the client device self configuring the DPU into the browser. This emulates the capability of a fully implemented DAN environment where DPU is automatically discovered and configured without end user intervention. For desktop and notebook computers using Internet Explorer, this can be done using the WININET library. However, such library does not exist for Pocket Explorer on the WinCE/Pocket PC platform. Alternative ways have to be explored.

3. Challenges for DPU Prototype

The objective of a DPU is to able to intercept the HTTP requests and response so that it can perform content repurposing based on the client device category. Fundamentally, a DPU is an intelligent HTTP proxy that is able to automatically transform, on the fly, the contents that are suitable for a requesting client device based on the device capability. Although there are many open source HTTP proxies available that we could reuse, there are some challenges in the design and implementation of the DPU prototype. These include the discovery of the DPU location, dynamic download and

execute of DPU code to support specialized client devices, and the ability to support session based content repurposing policy.

In our implementation, the location of a DPU is not fixed to support a particular web server. Assigning a designed DPU close to a web server has its advantages. Having a dedicated DPU, as opposed to a general DPU to support multiple web servers, allows the DPU to be specialized to handle repurposing tasks that is tailored to the contents served by the web server. In addition, locating the DPU close to the web server will minimize unnecessary network traffic that cannot be processed by the client devices. In order to support the discovery a DPU for each web server, some lookup mechanism must be in place, which is similar to those used in the Jini architecture.

There are many devices available and it is impossible for the DPU to be able to support all these devices. Hence, the challenge is to allow the client to specify the software codes that can handle the repurposing task that is optimized for the capability of the client device. Dynamically, the client should be to request a DPU to download and execute it in real time.

Most HTTP proxies are designed to work with a single type of client, and in most cases a desktop or a notebook computer is assumed. This model is insufficient to support the task that a DPU is designed to execute. Some form of adaptation is required to make a HTTP proxy to be able to determine the device type of an incoming HTTP request and perform the appropriate repurposing task.

C. PROTOTYPE DESIGN

1. Design Overview

Based on the analysis of the requirements, we identified the three software modules required for the prototype. They are the client browser, the Lookup Server and the DPU.

The client browser provides the interface of the DAN framework to the end user/device. It captures the information about the end user and device, and provides them to the other agents in the DAN framework.

The Lookup Server helps to direct the client browser to the appropriate DPU. It serves as the intermediate link between the client browser and the DPU.

The DPU is the nerve centre of a DAN network. It performs the essential DAN functions (e.g., capability negotiation, content repurposing, retrieve policy and proxy codes) based on inputs from the client browser.

The three modules -- the client browser, the Lookup Server, and the DPU -- work together to provide the functionalities of the DAN framework. The flow of events and the exchange of messages between these entities are illustrated in the sequence diagram below.

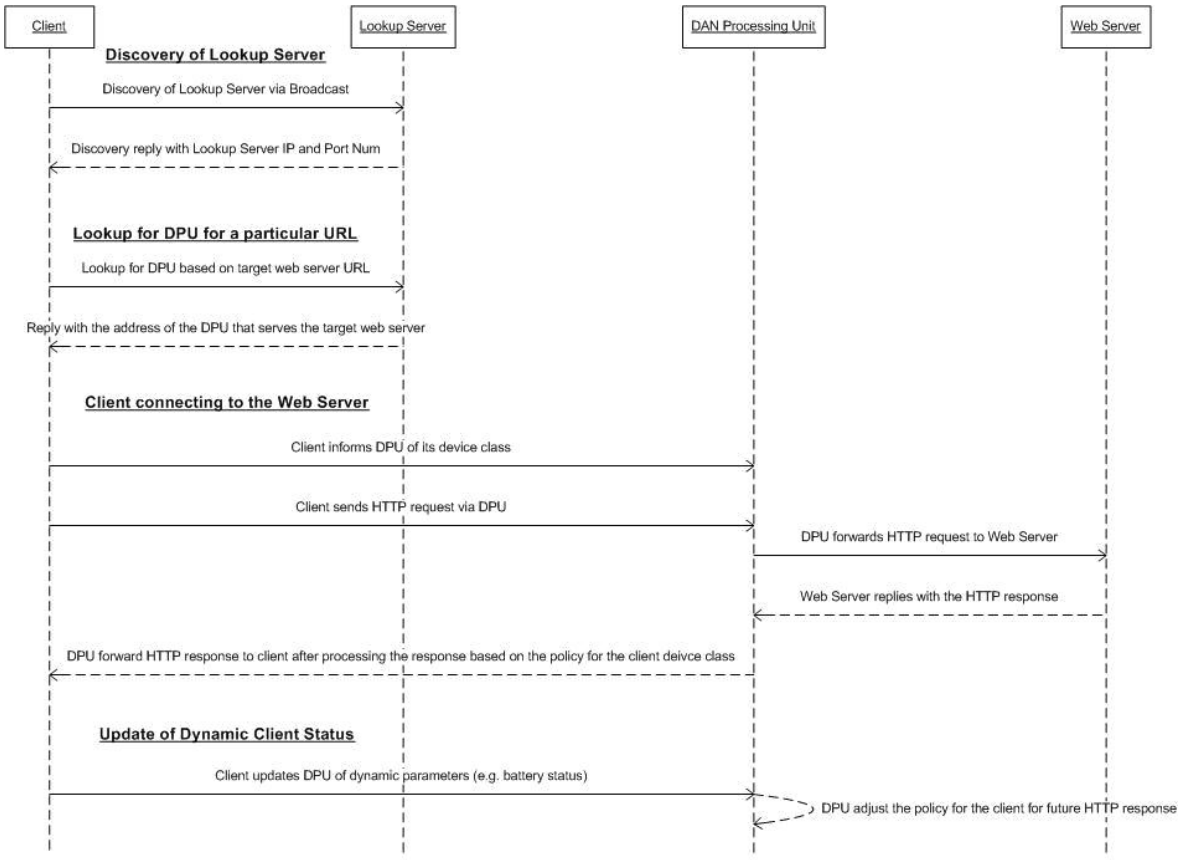

Figure 7. Sequence Diagram for DAN Framework Prototype

a. Lookup Server Discovery

When the client browser is launched, it will try to discover any Lookup Server in the proximity by broadcasting a Discovery Message. When a Lookup Server receives this message, it responds by sending a corresponding reply message with its IP address and port number.

The client browser will use the information in the reply to configure its Lookup Server settings. Subsequently, it will to use this Lookup Server whenever it needs to look up for a DPU corresponding to the website it tries to navigate to. In the event when multiple Lookup Servers respond (i.e., there are more than one Lookup Server in the proximity), the client browser will configure itself to the first Lookup Server response message it receives.

b. Lookup for DPU for a Particular URL

Before the client browser can navigate to the website specified by the user, it needs to establish a communication session with the DPU that corresponds to this particular website. The client device uses this session to provide administrative information to the DPU, such as its static device profile, user preferences and processing policy. It also uses this session to update the DPU with its dynamic profile such as battery status. The client browser uses the Lookup Server to find the address of the DPU for the web server it is targeting. This information can be derived from the URL entered by the user.

c. Client Connects to Web Server

Before the client sends the HTTP request, it first sends its static device profile, user preferences and processing policy to the DPU identified earlier. This information will be used by the DPU to process the contents returned by the web server. With all the settings in place, the client browser finally sends out the HTTP request to the DPU, which in turn, forward it to the web server.

The web server receives and processes the HTTP request like any regular request. This ensures that the DAN framework will work with all existing web servers. The web server then returns the requested content to the DPU.

Upon receiving the content, the DPU will perform capability matching, and repurpose the content whenever necessary. The repurposed content is then returned to the requesting client. As a result of the capability matching and content repurposing, the client browser is able to use and utilize every bit of the returned content. Thus, there is better utilization of resources for both the client device that receives and displays the content, and the network that delivers the content.

d. Update of Dynamic Client Status

Besides the static device information, the client browser will also update the DPU on its dynamic status such as the battery level. The DPU will re-apply the rules accordingly. This enables the DAN framework to response to changing operational climate.

2. Communication Messages

In order for the DAN client to communicate with a DPU, a new protocol is developed. It facilitates the synchronization of events and activities between the client and the DPU.

The information exchanges between a DAN client and DPU are based on American Standard Code for Information Interchange (ASCII) text messages with each field delimited by a comma. Below is a list of messages used in the DAN protocol.

a. Discovery Message

The Discovery Message is used by a DAN client to discover a Lookup Server as well as a Lookup Server to response to a client lookup request.

Message Fields		
Field Name	**Values/Type**	**Description**
Message Number	10	Unique message identifier.
Message Type	0 = Discovery Or 1 = Reply	Denotes whether this message is sent by a client for the discovery of a Lookup Server or a response sent by the Lookup Server in response a client request.
Port Number	0 – 65535	The port number field is used by both a client and a DPU to denote the port to listen for incoming messages.
Server Name	ASCII Text	The Internet Protocol (IP) address of the lookup server. This field is not required for a discovery.

Table 3. Message Fields for the Discovery Message

b. DPU Lookup Message

With the location of the Lookup Server known, a client uses the DPU Lookup Message to query a Lookup Server for the location of a DPU that supports a particular website.

Message Fields		
Field Name	**Values/Type**	**Description**
Message Number	100	Unique message identifier.
Target Web Server's URL	ASCII Text	The URL of web server that the client is trying to connect.
Client Device Type	0 = Desktop 1= Notebook 2= PDA 3 = Cell Phone	The broad category that a client device belongs to.
Client Device Model	ASCII Text	The model number of a client device.

Table 4. Message Fields for the DPU Lookup Message

c. DPU Update Message

The DPU Update Message is used by a Lookup Server in response to a client's request for a DPU lookup. It is also used by the DPU to inform a client of location and port number of the DPU that is hosting the specialized codes.

Message Fields		
Field Name	**Values/Type**	**Description**
Message Number	200	Unique message identifier.
Message Source	0 = Lookup Server 1 = DPU	Denotes the originator of this message.
Server Name	ASCII Text	The Internet Protocol (IP) address of a DPU.
Port Number	0 – 65535	The port number that a DPU is listening to.

Table 5. Message Fields for the DPU Update Message

d. Activate DPU Message

A DAN client uses the Activate DPU Message to inform a DPU of its device type and model. The DAN client can also request the DPU to download a specialized code host it at the DPU.

Message Fields		
Field Name	**Values/Type**	**Description**
Message Number	300	Unique message identifier.
Specialized Code's URL	ASCII Text	The URL of specialized code that a client requests for a DPU to download and execute.
Client Device Type	0 = Desktop 1 = Notebook 2 = PDA 3 = Cell Phone	The broad category that a client device belongs to.
Client Device Model	ASCII Text	The model number of a client device.

Table 6. Message Fields for the Activate DPU Message

e. Status Update Message

The Status Update Message is used for a client to update a DPU about its dynamic parameters. Currently, only the battery status is implemented.

Message Fields		
Field Name	**Values/Type**	**Description**
Message Number	300	Unique message identifier.
Battery Status	0 – 100	The percentage of power left for a client device
Port Number	0 – 65535	The port number that a client is listening to.
Client Device Type	0 = Desktop 1 = Notebook 2 = PDA 3 = Cell Phone	The broad category that a client device belongs to.
Client Device Model	ASCII Text	The model number of a client device.

Table 7. Message Fields for the Status Update Message

3. Client Design

This section discusses the design of the client browser. With the exception of the `WininetMgr` class, the client browser is implemented using C# (C Sharp). Based on the Microsoft .Net Framework, the client browser program is first compiled as Microsoft Intermediate Language (MSIL), which is a processor-independent instruction codes. Such programs are also known as managed codes. During runtime, the Common Language Runtime of the .Net Framework converts the program in MSIL to native codes and executes on the targeted operating system.

a. DocBrowser Class

The purpose of this class is to provide a customized Internet browser that incorporates DAN functionalities. It can be used to browse websites in the World Wide Web just like a regular web browser. In addition, user can turn on the DAN mode to use the built-in DAN functionalities when browsing the Internet. For the purpose of this

study, we also developed a simulation feature to that can used to simulate the browser's platform and battery status. The key attributes of the `DocBrowser` class is described in the Table 8.

Attribute Name	Description
`axBrowser1`	This is a Web Browser control that is used to navigate the World Wide Web and display web pages. It is similar to the one used in Internet Explorer for displaying web pages. Using this control allows us to intercept the event between the user entering the URL and clicking the "Go" button to start the navigation, and the browser initiating the HTTP protocols to navigate to the specified website. Such control is required for us to build in the DAN functionalities.
`FrmConnection`	This Form class is used for reading and setting the named proxy settings.
`menuOptions`	This menu option is a scaled down version of the "Internet Options" found in the Internet Explorer. It allows user to configure the proxy settings manually.
`menuMode`	This menu option allows user to set the browser to operate in the normal or DAN mode.
`menuBatt`	This menu option allows user to simulate the client device battery status. The available options are "High", "Moderate" and "Low".
`menuDevice`	This menu option allows user to simulate the device class of the client device. The available options are "Desktop", "Notebook", "PDA" and "Cellphone".

Table 8. Key Attributes of the DocBrowser Class

b. DiscoveryMgr Class

The purpose of this class is to search for the Lookup Server and the DPU, and to update the DPU on client device's dynamic profile. The key attributes of the `DiscoveryMgr` class is described in the Table 9.

Attribute Name	Description
`CommsMgr`	This class is used to send and receive messages with the DAN framework.
`FindLookupServer()`	This method broadcasts a probing message to look for the Lookup Server.
`FindPDU()`	This method queries the Lookup Server for the DPU for a given URL.
`UpdateBatteryStatus()`	This method updates the current DPU on the current battery status of the client device.
`UpdateDeviceClass()`	This method updates the current DPU on the device class of the client device. This method is used for simulation purpose only.
`OnDiscoveredDPU`	This event will trigger when a DPU is found. It will pass the DPU address to the event handler.

Table 9. Key Attributes of the DiscoveryMgr Class

c. WininetMgr Class

The class provides an abstraction of the WININET library relevant for the DAN framework. The actual working codes is programmed in C++ and compiled as a dynamic link library (i.e., WininetMgr.dll). This class provides an interface to the WininetMgr.dll, which runs in the unmanaged code realm. The key attributes of the `WininetMgr` class are described in Table 10.

Attribute Name	Description
`EnableProxy()`	This method sets the proxy to be used for the browser. It accepts the proxy IP address and port number. Since Internet Explorer also uses the same library (i.e., WININET library), the proxy settings done here will also affect other instances of the Internet Explorer on the device.
`DisableProxy()`	This method disables the proxy settings for the browser, and the browser will connect to the web server directly. Like `EnableProxy`, it settings will also affect other Internet Explorer instances on the same device.
`IsProxyUsed()`	This method checks the registry if the browser is configured to use named proxy. It returns true if named proxy is used, otherwise it returns false.
`GetProxyAddr()`	This method reads the proxy settings in the registry and returns the address of the named proxy. The proxy address returned consists of the IP address and port number.

Table 10. Key Attributes of the WininetMgr Class

d. FrmConnection Class

The purpose of this class is to allow user to read or set the named proxy settings manually if he wishes to override the proxy settings discovered by the client browser in DAN mode. It made use of the `WininetMgr` class to read and set the named proxy settings into the registry.

The key attributes of the `FrmConnection` class is described in the Table 11.

Attribute Name	Description
`WininetMgr`	The class is used to read and set the named proxy settings.

Table 11. Key Attributes of the FrmConnection Class

4. Lookup Server Design

The Lookup Server is the entity that facilitates the dynamic and loosely-coupled interaction between a DAN client and a DPU. The location of a Lookup Server is not made known to a client; instead the client has to send a broadcast message to discover the Lookup Server, which is listening for discovery messages at port 16800. In terms of the discovery process, this is very similar to a Dynamic Host Configuration Protocol (DHCP) client discovering the server to obtain its IP address [7].

The Lookup Server also maintains a list of web sites and the corresponding DPUs that are dedicated to service them. Upon startup, the Lookup Server reads an eXtensible Markup Language (XML) file, DpuProxyLookup.xml, that provides persistent storage for this mapping of web sites and DPU related information. The Document Type Definition (DTD) for DpuProxyLookup.xml is shown below. Based on this definition, we can see that DpuProxyLookup.xml contains one or more DpuProxy items and each DpuProxy item is made of up a web server URL, its IP address and the port number that the DPU will be listening. It is assumed that in a full Lookup Server implementation, the DpuProxyLookup.xml file will be generated by additional processes associated with the Lookup Server.

```
<!ELEMENT DpuProxyLookup (DpuProxy)*>

<!ELEMENT DpuProxy (weburl,ipaddr,port)>

<!ELEMENT weburl (#PCDATA)>

<!ELEMENT ipaddr (#PCDATA)>

<!ELEMENT port (#PCDATA)>
```

Figure 8. DTD for DpuProxyLookup.xml

The Lookup Server is implemented using Java 2 Platform Standard Edition version 5.0 (J2SE5.0). The main logic of the Lookup Server is embedded in the `LookupServer` class. It is supported by the `CommsManager` class and the `ServerInfo` class. The following are the detailed descriptions of the above mentioned classes.

a. CommsManager Class

The `CommsManager` class provides the functionality for an application to send and receive messages via User Datagram Protocol (UDP). The `CommsManager` creates a UDP socket that is used for both sending and receiving of messages. To send a message, the member method `sendUDP()` or `sendBroadcast()` is used and `CommsManager` will take care of the rest to call the appropriate methods in the `java.net` package to send the message.

To handle the receipt of messages, `CommsManager` will first create an `UDPMesssageReceiver` object (inherits from `java.lang.Thread`), which will spawn a thread to listen for incoming UDP message. This is an important step as `DatagramSocket.listen()` is a blocking call (i.e.; it will cause the software to remain dormant until a new message is received at the UDP socket). Hence, without the `UDPMesssageReceiver` object, the application that uses `CommsManager` will "hang" until a new message is received. Upon receiving a UDP message, `UDPMesssageReceiver` will call `onReceive()` in the `ReceivedMessageHandler` interface. The class using the `CommsManager` will have to provide the implementation for the `onReceive()` to handle the specifics of how to handle the received messages. The key attributes of the `CommsManager` class is described in Table 12.

Attribute Name	Description
`UDPMesssageReceiver`	The class is used to handle received messages for the `CommsManager`. Its function is described in the few paragraphs above.
`StartUDPReceiver()`	This method is used to activate the `UDPMessageReceiver` to listen to incoming messages.
`SendUDP()`	This method is used to send a UDP message to another node by providing the destination IP address and port number.
`SendBroadcast()`	This method is used to send a broadcast message to a particular port number.

Table 12. Key Attributes of the CommsManager Class

b. ServerInfo Class

The `ServerInfo` class represents each record for the mapping of a website URL to its corresponding DPU related information (i.e. its IP address and port number). The key attributes of the `ServerInfo` class is described in Table 13.

Attribute Name	Description
`getServerName()`	These methods retrieve and set the value of the `serverName` attribute (URL of the server) of the `ServerInfo` class respectively.
`setServerName()`	
`getServerAddress()`	These methods retrieve and set the value of the `serverAddr` attribute of the `ServerInfo` class respectively.
`setServerAddress()`	
`getPort()`	These methods retrieve and set the value of the `port` attribute of the `ServerInfo` class respectively.
`setPort()`	

Table 13. Key Attributes of the ServerInfo Class

c. LookupServer Class

The `LookupServer` class implements the logic and behavior of the Lookup Server. During startup, the constructor of the `LookupServer` class reads from a data file, DpuProxyLookup.xml, to build up a lookup table in memory via the `initializeDpuList()` method. This lookup table is implemented using a Hashtable object with the key as the URL of a web server and the value is a `ServerInfo` object with the corresponding DPU information that supports the web server. The Hashtable is used as the data structure as it allows quick reference to the DPU information when a client requests for it by sending the URL of the web site that the client wishes to connect. In addition, the constructor also creates an instance of the `CommsManager` class and starts the `UDPMesssageReceiver` object to listen at port 16800 for incoming messages.

As mentioned above, the `LookupServer` class also implements the `ReceivedMessageHandler` interface by providing the implementation of the `onReceive()` method. This method is called upon when the `UDPMessageReceiver` receives an incoming message at port 16800. The key attributes of the `LookupServer` class is described in Table 14.

Attribute Name	Description
`main()`	This method provides the entry point to the Lookup Server application. It starts the application by creating an instance of the `LookupServer` class.
`InitializeDpuList()`	This method is called to populate the Hashtable with the DPU related information stored in the DpuProxyLookup.xml data file.
`OnReceive()`	This method handles the behavior of the Lookup Server when an incoming message is received. Responses to messages are as follows: ▪ When it receives a Discovery Message (Type = Discover), it will reply to the client with another Discovery Message (Type = Reply) with the contents of the Lookup Server's IP address and port number. ▪ When a DPU Lookup Message is received, the Hashtable of DPU information is consulted to check whether it contains an entry based on the URL of the web site sent in the received message. If it exists, it will response with a DPU Update Message with the supporting DPU's IP address and port number. Otherwise, it will send the DPU Update Message with IP address as "0.0.0.0" and port number as -1 to denote that the Lookup Server does not know of the DPU that is supporting the requested web site.

Table 14. Key Attributes of the LookupServer Class

5. DPU Design

The DPU consists of two main components, namely a DAN protocol handler and a web proxy. The DAN protocol handler manages the message exchanges with a DAN client and determines the policy used for content repurposing based on the client device type. The web proxy intercepts HTTP requests and responses and performs the task of content repurposing based on the determined policy.

In this prototype, the web proxy is based on the RabbIT Web Proxy version 2.0.35 [8]. It is a Java based web proxy that complies with HTTP version 1.1. Several

modifications are made to the RabbIT Web Proxy source codes to enable it to perform content repurposing on images. Details on the modifications are documented in Section C, Prototype Implementation.

The policies for content repurposing focus only on images. For the purpose of this prototype, we have three policies, namely the Full Image, Low Resolution Image and No Image policies. Based on the device type, the DAN protocol handler will determine the policy using the following mapping.

Device Type	Policy
Desktop, Notebook	Full Image
PDA	Low Resolution Image
Cell Phone	No Image

Table 15. Mapping for Device Type to Policy

In addition, dynamic parameters (such as battery level) from the client devices are also used as factors to determine policies. For example, if the battery level is lower than the threshold, the following policy for each client devices type will be enforced.

Device Type	Policy
Desktop, Notebook	Low Resolution Image
PDA	No Image
Cell Phone	No Image

Table 16. Policy for Device Type when the Battery Level is Below Threshold

Similar to the Lookup Server, the DAN protocol handler is implemented using J2SE5.0. It is implemented as part of the `DanProcessingUnit` class. In addition, the `DanProcessingUnit` class also loads and executes the RabbIT Web Proxy Java classes upon startup. The following are the detailed descriptions of the classes involved, which include the `JarClassLoader` class, `PolicyInfo` class and `PolicyManager` class.

a. JarClassLoader Class

The `JarClassLoader` class handles the loading and execution of a Java Archive (JAR) file. It allows the loading of remote Java classes and resources that are packed in a JAR file and stored in a remote server.

The source codes used in this class are based on the example used in the lesson on using JAR-related APIs from the Java Tutorial [9]. It is extended to include a function to extract a file from JAR file. The key attributes of the `JarClassLoader` class is described in Table 17.

Attribute Name	Description
`GetMainClassName()`	This method determines the class from the classes packed in a JAR file that contains the static `main()` function. It uses the information from the manifest file that is packed together with the JAR to determine this.
`InvokeClass()`	This method invokes the static `main()` function by supplying the name of the main class and an array of arguments that is recognized by the `main()` function.
`ExtractFileFromJar()`	This method extracts a file that is packed in a JAR file.

Table 17. Key Attributes of the JarClassLoader Class

b. PolicyInfo Class

The `PolicyInfo` class represents the policy that will be applied for a particular client device. It stores the policy, in the form of a text string, as well as the client IP address. The key attributes of the `PolicyInfo` class is described in Table 18.

Attribute Name	Description
`getPolicy()`	These methods retrieve and set the value of the `policy` attribute of the `PolicyInfo` class respectively.
`setPolicy()`	
`getClientAddress()`	These methods retrieve and set the value of the `clientAddr` attribute of the `PolicyInfo` class respectively.
`setClientAddress()`	

Table 18. Key Attributes of the PolicyInfo Class

c. PolicyManager Class

The `PolicyManager` class manages the content repurposing policies for all client devices that are connected to the DPU. The `PolicyManager` maintains a policy table of client IP addresses and the corresponding policies. This table is updated whenever the client sends the Activate DPU Message to the DPU to the DPU Admin Port 19800. Based on the client device type, the `PolicyManger` will assign the appropriate policy (based on the mapping in Table 15) and update the policy table accordingly. In addition, the Status Update Message also triggers the `PolicyManager` to update the policy table. In this case, the client device updates its battery status and if it falls below at pre-defined threshold (currently it is set at 40%), the `PolicyManager` change the policy for the client based on the mapping defined on Table 16.

The `PolicyManager` also plays a key role in conveying the policy information to the RabbIT Web Proxy. As the RabbIT Web Proxy is running in another process, some form of inter process communications means is required to inform the RabbIT Web Proxy of the policy for the clients that may connect to it. Instead of setting up an extensive infrastructure for inter process communications, we used an XML file to exchange the policy related information between the `PolicyManager` and the RabbIT Web Proxy. For each update to the policy table, the `PolicyManager` will generate a similar set of information and write to the policies.xml file. When the RabbIT Web proxy needs to decide on the policy to use, it will read the policies.xml file to extract the required information. The DTD for the policies.xml file is shown below. The policies.xml file contains one or more policy items and each policy item is made of up a client IP address and the description of the policy.

```
<!ELEMENT policies (policy)*>

<!ELEMENT policy (address,description)>

<!ELEMENT address (#PCDATA)>

<!ELEMENT description (#PCDATA)>
```

Figure 9. DTD for Policies.xml

The key attributes of the `PolicyManager` class is described in Table 19.

Attribute Name	Description
`UpdatePolicy()`	This method updates the policy table using the client device type and IP address as the input.
`UpdateDynamicPolicy()`	This method determines and updates the policy table based on the dynamic parameters such as the battery level of the client device.
`WritePoliciesToFile()`	This method copies the policy table and writes it to the policies.xml file.

Table 19. Key Attributes of the PolicyManager Class

d. DanProcessingUnit Class

The `DanProcessingUnit` class is the main class for the DPU. It contains the static `main()` function that bootstraps the application. The main() function initializes the classes that the `DanProcessingUnit` references (`PolicyManager` class and `CommsManager` class) through its constructor and starts to load and execute the RabbIT Web Proxy. Once the RabbIT Web Proxy is loaded, the URL of the RabbIT Web Proxy codes and the port number for the proxy (default port number is 9666) are added to a data structure called the active proxy table. This table keeps track of all instances of RabbIT Web Proxy that are active in memory.

At any given time, there is at least one instance of the RabbIT Web Proxy running. Client devices can, through the Activate DPU Message, request the DPU to download a copy of specialized RabbIT codes to handle the specifics that are unique to a particular class of devices. The URL of the code is specified by the client in the Specialized Code's URL field of the Activate DPU Message. On receiving this message, the `DanProcessingUnit` class will utilize the `JarClassLoader` class to download the codes from the specified URL. If the URL is valid, the `JarClassLoader` class will execute the downloaded codes, causing another instance of the RabbIT to execute and listen on another port for web requests and responses. Once the new instance is executing, the `DanProcessingUnit` class will add another entry to the active proxy table with the URL of the specialized codes and the new port number as inputs. In

addition, the `DanProcessingUnit` class will send a DPU Update Message to inform the client to connect to this new RabbIT Web Proxy instance for its subsequent web requests. However, this could be a potential security flaw as the client can send a URL that contains malicious codes and cause the DPU to download and execute such codes. Therefore, in the future implementation of the DPU, security measures must put in place to prevent this from happening.

Similar to the Lookup Server, the `DanProcessingUnit` class utilizes the `CommsManager` class to send and receive messages from client devices. It also implements the `ReceivedMessageHandler` interface by providing the implementation of the `onReceive()` method. This method is invoked when an incoming message arrives at DPU Admin Port 19800. The key attributes of the `DanProcessingUnit` class is described in Table 20.

Attribute Name	Description
`main()`	This method provides the entry point to the `DanProcessingUnit` application. It starts the application by creating an instance of the `DanProcessingUnit` class.
`loadApp()`	This method activates the `JarClassLoader` class to download and execute an application from the specified URL.
`addProxy()`	This method adds an entry to the active proxy table.
`onReceive()`	This method handles the behavior of the DPU when an incoming message is received. Responses to messages are as follows: a. When the Active DPU Message is received, it will check if the URL of the requested codes exists in the active proxy table. If it does not exist in the active proxy table, the codes will be downloaded from the URL and executed. Next, the `DanProcessingUnit` class will send a DPU Update Message to inform the client to connect to this new RabbIT Web Proxy instance for its subsequent web requests. Once this is completed, the `PolicyManager` is activated to determine the policy for the client device. b. When the DPU receives the Status Update Message, it will activate the `PolicyManager` to update the policy based on the dynamic parameter sent by the client device.

Table 20. Key Attributes of the DanProcessingUnit Class

D. IMPLEMENTATION DETAILS

1. Implementing DAN Browser

As a client to a DAN network, we developed a customized web browser, we call the DAN browser, that performs various functions required by the DAN framework. In order to develop the prototype to facilitate the demonstration of the DAN framework quickly, we tried to reuse existing software components as much as possible. Therefore, instead of developing a web browser from scratch, we make use of the Web Browser control that is used in the Internet Explorer. We have also considered using open source browsers, but found that the learning curve will take too long to fit our schedule.

For the prototype, we used the Microsoft .Net Framework (specifically, C# language) to develop the client application (i.e., DAN browser). As such framework works on managed code, the corresponding Windows Forms can only host Windows Forms controls (i.e., derived classes of `System.Windows.Forms.Control`). Unfortunately, the Web Browser is an ActiveX control that is developed in earlier platforms using unmanaged code. In order to use the control in a managed code environment, it will need to be embedded in a Windows Forms control class (i.e., `System.Windows.Forms.AxHost` class). This wrapper class acts as a host for the ActiveX control so that it exposes a similar interface as a regular control class.

The Web Browser control can hence be converted to a regular control class by using the control wrapper toolkit, Aximp.exe tool, which is included in the .Net Framework SDK. To do this, we need to convert the entire type library of the Web Browser control (i.e., shdocvw.dll) using the command below. The output of Aximp.exe is a set of binary files that contains the metadata and control implementation for the types defined within the original type library. In this case, it is easier to assume that the converted control is contained in the newly produced type library file Axshdocvw.dll.

```
aximp  c:\winnt\system32\shdocvw.dll
```

In order to use the converted control in the windows form, we need to add a reference to the newly created type library, and initialize an instance of the control. The code excerpt is given below.

```
....
using AxSHDocVw;
....
namespace DANBrowser
{
      private AxSHDocVw.AxWebBrowser axWebBrowser1;

      ....
      this.axWebBrowser1 = new AxSHDocVw.AxWebBrowser();

      ....
      ((System.ComponentModel.ISupportInitialize)
            (this.axWebBrowser1)).BeginInit();

      ....
      this.axWebBrowser1.Enabled = true;
      this.axWebBrowser1.Location = new System.Drawing.Point(16, 24);
      this.axWebBrowser1.OcxState =
            ((System.Windows.Forms.AxHost.State)
            (resources.GetObject("axWebBrowser1.OcxState")));
      this.axWebBrowser1.Size = new System.Drawing.Size(528, 150);
      this.axWebBrowser1.TabIndex = 0;

      ....
      this.Controls.Add(this.axWebBrowser1);

      ....
      ((System.ComponentModel.ISupportInitialize)
            (this.axWebBrowser1)).EndInit();

      ....

}
```

Finally, the Web Browser control can be used to navigate to any website. The code excerpt is given below.

```
      ....
      //default arguments for Navigate method.
      object arg1 = 0; object arg2 = ""; object arg3 = ""; object arg4
= "";

      //navigate to URL/FilePath entered
      this.axWebBrowser1.Navigate(textURL.Text, ref arg1, ref arg2, ref
            arg3, ref arg4);
      ....
```

The Web Browser exposes a comprehensive range of events. These events are useful to moderate the behavior of the browser, as well as checking the status of the navigation. The following code excerpt shows the events that are used in this implementation.

```
      ....
      //Add Handler for Title Change event
      this.axWebBrowser1.TitleChange += new
AxSHDocVw.DWebBrowserEvents2_TitleChangeEventHandler(axWebBrowser1_Titl
eChange);

      //Add Handler for StatusBar change event
      this.axWebBrowser1.StatusTextChange += new
DWebBrowserEvents2_StatusTextChangeEventHandler(axWebBrowser1_StatusTex
tChange);

      //Add Handler for Progress Update event
      this.axWebBrowser1.ProgressChange += new
DWebBrowserEvents2_ProgressChangeEventHandler(axWebBrowser1_ProgressCha
nge);

//Add Handler for Handle Created event
//To interact with the newly created Browser control, so that we can
//use it to navigate to start page
      this.axWebBrowser1.HandleCreated += new
                  EventHandler(axWebBrowser1_HandleCreated);

      //Add Handler for Before Navigate Event
      this.axWebBrowser1.BeforeNavigate2 += new
DWebBrowserEvents2_BeforeNavigate2EventHandler(axWebBrowser1_BeforeNavi
gate2);
      ....
private void axWebBrowser1_HandleCreated(object sender, EventArgs args)
{
      try
      {
            //Navigate to Start Page
            this.buttonGo_Click(buttonGo.Text, EventArgs.Empty);

            //Remove EventHandler since it is no longer needed
            this.axWebBrowser1.HandleCreated -= new
                  EventHandler(axWebBrowser1_HandleCreated);
      }
      catch (Exception e)
      {
            this.ShowErrorMessage(e);
      }
}

private void axWebBrowser1_TitleChange(object sender,
                        DWebBrowserEvents2_TitleChangeEvent arg)
{
      //Display the Web Browser title on status bar
      this.Text = "DAN Browser - " + arg.text;
```

```
}

private void axWebBrowser1_StatusTextChange(object sender,
                        DWebBrowserEvents2_StatusTextChangeEvent e)
{
      if ((e.text == null) || (e.text == ""))
            updateStatusText("Done");
      else
            updateStatusText(e.text);
}

private void axWebBrowser1_ProgressChange(object sender,
                        DWebBrowserEvents2_ProgressChangeEvent e)
{
      if (e.progress < e.progressMax)
      {
            double progress = ((double)e.progress /
                  (double)e.progressMax) * 100.0;
            this.statusProgress.Text = ((int)progress).ToString() + "
                                    percent";
      }
      else //already completed
            this.statusProgress.Text = "Completed";
}

private void axWebBrowser1_BeforeNavigate2(object sender,
                        DWebBrowserEvents2_BeforeNavigate2Event e)
{
      if (e.pDisp == axWebBrowser1)
            MessageBox.Show("Before Navigate Event - " + e.uRL, "hi",
                              MessageBoxButtons.OK,
                              MessageBoxIcon.Information);
      else
      {
            MessageBox.Show("Block PopUp " + e.uRL, "hi",
                              MessageBoxButtons.OK,
                              MessageBoxIcon.Information);
            e.cancel = false;
      }
}
```

2. Configuring Proxy for DAN Browser

Since the DAN browser uses the Web Browser control, we use the WININET API to access and modify the registry settings for the browser. The Web Browser control shares the same registry entries as the mainstream Internet Explorer. Therefore, it should be noted that any changes to the settings performed through the DAN browser will affect any Internet Explorer running on the host, and vice versa. The APIs used to configure the

proxy settings for the DAN browser, written in C++, are shown in the header file listing below.

```
//Header File

#ifndef INTERNET_OPTION_H
#define INTERNET_OPTION_H

#include "stdafx.h"
#include <windows.h>  //required to avoid compilation error with
wininet.h
#include <wininet.h>
//#include <iostream>

//using namespace std;

class InternetOption {

private:

      unsigned long nSize; // size of INTERNET_PER_CONN_OPTION_LIST;

      INTERNET_PER_CONN_OPTION_LIST queryList; //store the various
options (i.e., queryOptions)to be queried.
      INTERNET_PER_CONN_OPTION queryOption[5]; //store settings
information

      bool queryInternetSettings(); //to refresh the values in
queryOptions
      unsigned long getConnectionFlag();

public:
      InternetOption();
      ~InternetOption();

      bool enableDirectProxy(TCHAR proxyAddr[], TCHAR bypassAddr[] );
      bool disableDirectProxy();

      bool isNamedProxyConnection();
      LPTSTR getByPassURL();
      LPTSTR getProxyIP();
      void showCurrentConnSettings(); //get the connection settings -
may need enum values
};
#endif
```

Like the Web Browser control, we need to perform some conversion to allow the WININET API related codes to run the managed environment. To do this, we wrapped these codes in the dynamic link library, and later access the exposed functions through the Platform Invoke feature provided with .Net Framework. The following code excerpt shows how we wrapped the codes in a dynamic link library.

```
#ifdef WININETMGR_EXPORTS
#define WININETMGR_API extern "C" __declspec(dllexport)
#else
#define WININETMGR_API _declspec(dllimport)
#endif

//Includes...
#include "InternetOption.h"

WININETMGR_API    bool  enableProxy(char  proxyAddr[],  char  bypassAddr[]
);
WININETMGR_API    bool disableProxy(void);
WININETMGR_API    bool getProxyAddr(LPTSTR proxyAddr);
WININETMGR_API    int isProxyUsed(void);

#include "stdafx.h"
#include "WinINETMgr.h"
#include "InternetOption.h"
#include <string>

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                               )
{
      switch (ul_reason_for_call)
      {
      case DLL_PROCESS_ATTACH:
      case DLL_THREAD_ATTACH:
      case DLL_THREAD_DETACH:
      case DLL_PROCESS_DETACH:
            break;
      }
    return TRUE;
}


WININETMGR_API bool enableProxy(char* proxyAddr, char* bypassAddr)
{
      InternetOption inet;
      return inet.enableDirectProxy(proxyAddr, bypassAddr);
}

WININETMGR_API bool disableProxy(void)
{
      InternetOption inet;
      return inet.disableDirectProxy();
}

WININETMGR_API bool getProxyAddr(LPTSTR proxyAddr)
{
      InternetOption inet;
      int len = strlen((char *) inet.getProxyIP());
      lstrcpyn(proxyAddr, inet.getProxyIP(), len + 1);

      return true;
}
```

```
WININETMGR_API int isProxyUsed(void)
{
	InternetOption inet;
	if (inet.isNamedProxyConnection())
		return 1;
	else
		return 0 ;
}
```

Finally, to access the WININET API functions from the DAN browser form, we created a class (i.e., `WinInetMgr`) to import the functions from the DLL. The following code excerpt is extracted from `WininetMgr` class.

```
using System.Runtime.InteropServices;
using System.Text;
using System;

namespace DANBrowser
{
	/// <summary>
	/// Summary description for WininetMgr.
	/// </summary>
	public class WininetMgr
	{
		[DllImport("WininetMgr.dll")]
		public static extern bool enableProxy(String proxyAddress,
                                                String localAddress);

		[DllImport("WininetMgr.dll")]
		public static extern bool disableProxy();


		[DllImport("WininetMgr.dll")]
		public static extern bool getProxyAddr(StringBuilder
                                                 proxyAddr);

		[DllImport("WininetMgr.dll")]
		public static extern int isProxyUsed();
	}
}
```

4. Modifications to the RabbIT Web Proxy

In order the support the requirements of a DPU, the following code changes were made to the original RabbIT Web Proxy source codes.

a. New Input Argument to Specify the Port for the RabbIT Web Proxy

The original RabbIT Web Proxy obtains the proxy port number from a configuration file. This presents a problem for the `DanProcessingUnit` class to execute another instance of the RabbIT Web Proxy as it will read from the same configuration file. Hence, there is a need to customize the `rabbit.proxy.Proxy` class from the `rabbit.proxy` package, which is the main class for the RabbIT Web Proxy. First, the `main()` function is modified to take in an addition argument "`-p`" or "`--port`" to specify the proxy port number. This value is stored in the static variable `port`.

```
public static void main(String[] args)
{
      ....
      ....
       else if (args[i].equals("-p") ||
                 args[i].equals("--port"))
       {
            i++;
            if (args.length > i)
            {
                  port = Integer.parseInt(args[i]);
            }
            else
            {
                  logError(FATAL,"No port specified");
                  System.exit(-1);
            }
       }
       ....
       ....
}
```

Next, the `openSocket()` function in the `Proxy` class is modified so that it will take in the value stored in the `port` variable if the port argument is specified when the RabbIT Web Proxy is executed. Otherwise, it will read from the configuration file to obtain the proxy port number.

```
protected static void openSocket()
{
	int tport;

	if (port == -1)
	{
		tport = Integer.parseInt(config.getProperty
			(Proxy.class.getName(), "port", "9666").trim());
	}
	else
		tport = port;

	try
	{
		port = tport;
		accepting = false;
		closeSocket();
		ss = new ServerSocket(port);
		accepting = true;
	}
	catch(IOException e)
	{
		logError(FATAL, "Failed to open serversocket on port " +
			port);
		System.exit(-1);
	}

}
```

b. New Image Handler Classes

The RabbIT Web Proxy uses the `rabbit.handler.ImageHandler` class to handle images downloaded from web servers. It utilizes an external application, ImageMagick [10], to reduce the quality of the image so as to minimize on the downloading time for images.

Several additions and modifications are made to the `rabbit.handler` package to support the three image policies. To handle the Full Image policy, the `rabbit.handler.ImageHandler` class is modified to disable the conversion routine. This modification takes place in the `setup()` function where the `doConvert` flag is set to false.

```
public static void setup(Properties prop)
{
	config = prop;
	doConvert = false;
}
```

A new class, `rabbit.handler.LowQualityImageHandler`, is created and added to the `rabbit.handler` package to support the Low Resolution Image policy. In terms of the code structure and the main routine, the `rabbit.handler.LowQualityImageHandler` is very similar to the `rabbit.handler.ImageHandler`. The difference is only in the `convertImage()` function. It makes use of the Java ImageIO classes to reduce to the quality of image instead of the ImageMagick application. The advantage of using ImageIO classes is that it eliminates the reliance of an external application that is executed in another process via the Java Runtime object. The codes for the `convertImage()` and the helper function `compressJpegFile()` are shown below.

```
protected void convertImage() throws IOException
{
	String qualityStr = config.getProperty("quality", STD_QUALITY);
	String entryName = Proxy.getCache().getEntryName(entry.getId(),
                                                    false);

	try
	{
		File oriFile = new File(entryName);
		convertedFile = new File(entryName + ".conv");
		compressJpegFile(new File(entryName),
                                convertedFile,
                                quality);

		lowQualitySize = convertedFile.length();
		response.setHeader("Content-Type", "image/jpeg");
		File oldEntry = new File(entryName);
		oldEntry.delete();

		if (convertedFile.renameTo (new File(entryName)))
			convertedFile = null;
	}
	finally
	{
		if (convertedFile != null)
		{
			convertedFile.delete();
		}
	}

	size = lowQualitySize;
	response.setHeader("Content-length", "" + size);
	con.setExtraInfo("imageratio:" + origSize + "/" + lowQualitySize
                    + "=" + ((float)lowQualitySize / origSize));

	contentstream.close();
	contentstream = new FileInputStream(entryName);
```

```
        convertedFile = null;
}

....
....

private void compressJpegFile(File infile,
                              File outfile,
                              float compressionQuality)
                              throws IOException, FileNotFoundException
{
        BufferedImage input = ImageIO.read(infile);

        // Get Writer and set compression
        Iterator iter = ImageIO.getImageWritersByFormatName("JPG");
        if (iter.hasNext())
        {
                ImageWriter writer = (ImageWriter)iter.next();
                ImageWriteParam iwp = writer.getDefaultWriteParam();
                iwp.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);
                iwp.setCompressionQuality(compressionQuality);

                FileImageOutputStream output =
                                new FileImageOutputStream(outfile);
                writer.setOutput(output);
                IIOImage image = new IIOImage(input, null, null);
                writer.write(null, image, iwp);
                writer.dispose();
                output.close();
        }
}
```

In order to handle the client devices that use the No Image policy, the `rabbit.handler.BlockImageHandler` class is created. It is identical to the `rabbit.handler.LowQualityImageHandler` class with the exception of the `convertImage()` function. This function replaces all images with a blank image file. Below are the codes for the `convertImage()` and the helper function `copyJpegFile()`.

```
protected void convertImage() throws IOException
{
        String entryName = Proxy.getCache().getEntryName(entry.getId(),
                                                         false);
        String bimg = config.getProperty("blankimage", BLANK_IMAGE);

        try
        {
                File oldEntry = new File(entryName);
                oldEntry.delete();

                convertedFile = new File(bimg);

                lowQualitySize = convertedFile.length();
```

```
            response.setHeader("Content-Type", "image/jpeg");

            copyJpegFile(convertedFile,oldEntry);

      }
      catch(FileNotFoundException fnfe)
      {
            Proxy.logError(Proxy.ERROR, "Blank image -" + bimg +
                        "- not found, is your path correct?");
      }
      catch(IOException ioe)
      {
            Proxy.logError(Proxy.ERROR, "IO Error in copying JPEG file"
                                          + ioe);
      }

      size = lowQualitySize;
      response.setHeader("Content-length", "" + size);

      con.setExtraInfo("imageratio:" + origSize + "/"
                       + lowQualitySize + "="
                       + ((float)lowQualitySize / origSize));

      contentstream.close();
      contentstream = new FileInputStream(entryName);
      convertedFile = null;
}
....
....
private void copyJpegFile(File infile, File outfile)
                          throws IOException, FileNotFoundException
{
      BufferedImage input = ImageIO.read(infile);

      Iterator iter = ImageIO.getImageWritersByFormatName("JPG");
      if (iter.hasNext())
      {
            ImageWriter writer = (ImageWriter)iter.next();

            FileImageOutputStream output =
                           new FileImageOutputStream(outfile);
            writer.setOutput(output);
            writer.write(input);
            writer.dispose();
            output.close();
      }
}
```

c. Policy Handling

The `rabbit.proxy.Connection` class is responsible for the HTTP connection from the web proxy to a client device or a web server. Therefore, this is the class where the modification to handle polices takes place. As mentioned previously, the

PolicyManager class communicates the policies for each of the devices to the RabbIT Web Proxy via the policies.xml file. In order to support that, XML processing components are added to the `rabbit.proxy.Connection` class and they are initialized in the constructor by calling the `setupPolicyTable()` function.

```
private void setupPolicyTable()
{
	try
	{
		//setup XML components
		DocumentBuilderFactory factory =
				DocumentBuilderFactory.newInstance();
		factory.setValidating(true);
		factory.setIgnoringElementContentWhitespace(true);
		builder = factory.newDocumentBuilder();
		policiesTable = new Hashtable<String, PolicyInfo>();
	}
	catch(ParserConfigurationException pce)
	{
		Proxy.logError(Proxy.ERROR,
				"Problems initializing XML components..."
				+ pce);
	}
}
```

The `policiesTable` is a Hashtable that contains the IP address of a client device as the key and the corresponding policy to be used as the value. The `updatePolicyTable()` is added to the `rabbit.proxy.Connection` class to facilitate the reading of the policies.xml file and update its contents to the `policiesTable`.

```
private void updatePolicyTable(File infile)
{
	Element childElement;
	Node childNode;
	Text value;
	String pol,addr;

	try
	{
		policiesTable.clear();
		Document doc = builder.parse(infile);

		Element root = doc.getDocumentElement();
		NodeList children = root.getChildNodes();

		for(int i = 0; i < children.getLength(); i++)
		{
			childElement = (Element)children.item(i);

			childNode = childElement.getFirstChild();
```

```
                value = (Text)childNode.getFirstChild();
                addr = value.getData().trim();

                childNode = childNode.getNextSibling();
                value = (Text)childNode.getFirstChild();
                pol = value.getData().trim();

                policiesTable.put(addr, new PolicyInfo(pol,addr));

            }
        }
        catch(org.xml.sax.SAXException se)
        {
            Proxy.logError (Proxy.ERROR, "Problems parsing "
                                        + infile.getPath() + "..." +
                                        se);
        }
        catch(IOException ioe)
        {
                Proxy.logError (Proxy.ERROR, "Problems parsing "
                                            + infile.getPath() +
                                             "..." + ioe);
        }
}
```

When a HTTP request is received at the RabbIT Web Proxy, the `handleRequest()` function will be activated. The following codes are added to the `handleRequest()` function to incorporate the feature that process images differently based on the policy for a particular client device.

The code first checks if the client is requesting an image (jpeg, gif or png). If an image is requested, the IP address of the client device is obtained from the `socket` class. This IP address is then used as a key to query the `policiesTable` for the required policy for this particular client device. If a policy is available, the appropriate image handler function (either `ImageHandler` for Full Image policy, `LowQualityImageHandler` for Low Resolution policy or `BlockImageHandler` for No Image policy) will be eventually called to process the image.

```
public void handleRequest(HTTPHeader header)
{
        RequestHandler rh = new RequestHandler();
        ....
        ....
        // ok get the handler for it.
        if (rh.chandler == null) {
        String ct = rh.webheader.getHeader("Content-Type");
        if (ct != null)
            ct = ct.toLowerCase();
        if (getMayFilter() &&
```

```
            rh.webheader != null && ct != null)
      {
            if ((ct.equals("image/jpg"))
                  || (ct.equals("image/jpeg"))
                  || (ct.equals("image/gif"))
                  || ct.equals("image/png"))
            {
                  String clientAddr =
                        socket.getInetAddress().getHostAddress().trim();
                  System.out.println("Connection from " + clientAddr);

                  File policyFile = new File(polFile);
                  if (policyFile != null)
                        updatePolicyTable(policyFile);

                  if (policiesTable.containsKey(clientAddr))
                  {
                        rh.chandler =
                           (Class)Proxy.handlers.get(
                                 policiesTable.get(clientAddr).getPolicy());
                        System.out.println("Image Handler : "
                                           + rh.chandler.getName());

                  }
                  else
                  {
                        rh.chandler =
                           (Class)Proxy.handlers.get(ct.toLowerCase ());
                        System.out.println("Image Handler : "
                                          + rh.chandler.getName());
                  }

            }
            else
            {
                  rh.chandler =
                           (Class)Proxy.handlers.get(ct.toLowerCase ());

            }
      }
      ....
}
```

E. DEMONSTRATED CAPABILITY

We demonstrated the concept of the DAN framework using a prototype we developed as part of the thesis requirements. In the demonstration, we showed how the DAN framework can enhance the network by enabling it to differentiate the client devices so as to deliver the optimal content to them. In addition, it is able to respond appropriate to the device's dynamic profile as well.

1. Prototype Setup

The prototype demonstration was carried out in the Mobile and Wireless Devices lab at the Naval Postgraduate School. The setup for the demonstration consists of the DAN browser, the Lookup Server and the DPU. All these applications were hosted on separate desktop terminals in the lab. These terminals are inter-connected by an Ethernet local area network, which is linked to the internet.

The DAN browser is hosted on a Windows 2000 workstation. For the purpose of the demonstration, it can be simulated to be hosted on other platforms like a PDA or a cell phone. It was designed this way so to better demonstrate the capability of the DAN framework. Take the case where the host is simulated as a cell phone which cannot display images. If we use an actual cell phone for the demonstration, it will not display the images even if it receives the images from the DAN framework. This will not happen in the simulated cell phone, as the DAN browser is capable of displaying all the images it receives.

Similarly, the Lookup Server and DPU are also hosted on Windows 2000 workstations. It should be noted that the platform does not matter since both are written using Java, which can run on any operating systems that support Java Virtual Machine (JVM). Using the Windows platform for the demonstration is simply a matter of convenience. Both Lookup Server and DPU services must be started prior to the demonstration.

To show that the DAN framework does not require any changes on the part of the information server (i.e., web server), we used external web servers for the demonstration. We selected Google and Answers.com websites for this demonstration.

The prototype DAN framework is shown below. The client device hosts the DAN browser, and can be used to simulate as a cell phone, PDA or laptop. The Lookup server and DPU are hosted on separate terminals.

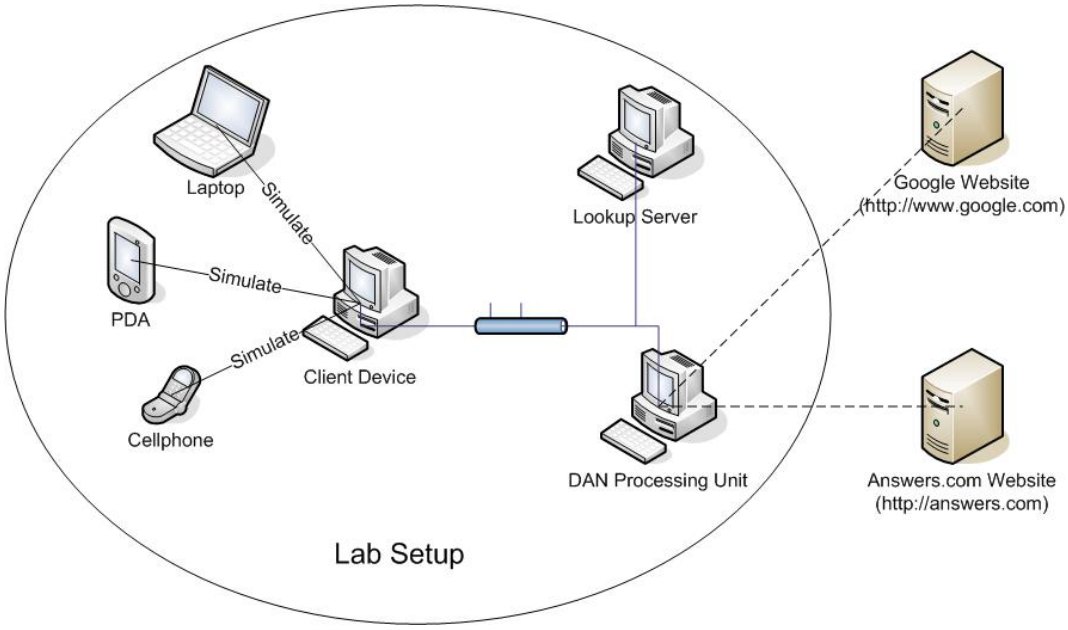

Figure 10. Prototype Setup

2. Prototype Demonstration

The demonstration of the DAN framework prototype is broken into two parts: 1) to demonstrate the ability to differentiate the end devices according to the static profile submitted (e.g., device class), and 2) to demonstrate the ability to adapt to the dynamic profile of the end device (e.g., battery status).

The DAN browser Graphical User Interface (GUI) (see figure 8) is shown below. It can operate in normal or DAN mode. In normal mode, it behaves like any regular browser. However, when DAN mode is switched on, it will interact with other agents in the DAN framework to ensure that the content it received is optimize for its capability and resources.

Figure 11. The DAN Browser

To facilitate the demonstration, the DAN browser may be simulated as the various device classes. The available options are Desktop, Notebook, PDA and Cell phone.

Figure 12. Options for Device Class

In addition, the DAN browser can also simulate the various battery levels to demonstrate the ability of the DAN framework to respond to changes in the device's dynamic profile.

Figure 13. Options for Battery Status

a. PDA in Normal Mode

The default setting for the DAN browser is to operate in the normal mode, like a regular web browser. The simulated platform is PDA. The browser is used to navigate to the Google website.

As expected, the browser displays the original webpage in its full entirety (see figure below), as what you would get operating in the regular network.

Figure 14. PDA in Normal Mode – Full Image

b. PDA in DAN Mode

Again, the browser is used to navigate to the Google website, except now The DAN browser is set to operate in the DAN mode. The web page is the same as before, except that the images are of lower resolution. The DAN framework is able to detect that the device is PDA and applied the appropriate policy to the content requested by the client. In this case, the DPU reduced the resolution of all the images associated with the returned content.

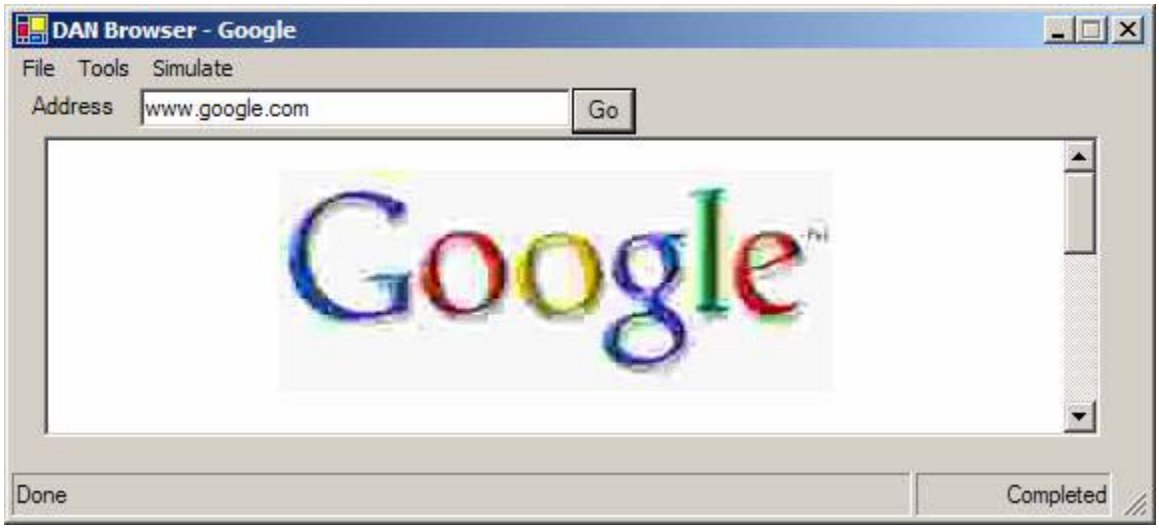

Figure 15. PDA in DAN Mode – Reduced Resolution Image

c. Cell Phone in DAN Mode

As before, the DAN browser is used to navigate to the Google website, but simulated to run on a cell phone. The result is that all the images are stripped from the web page displayed. The only reason this happen is because the DPU stripped all the images from the returned content (see Figure 13). Again, this demonstrated that ability of DPU to differentiate the static profile of the end device and customized the content accordingly.

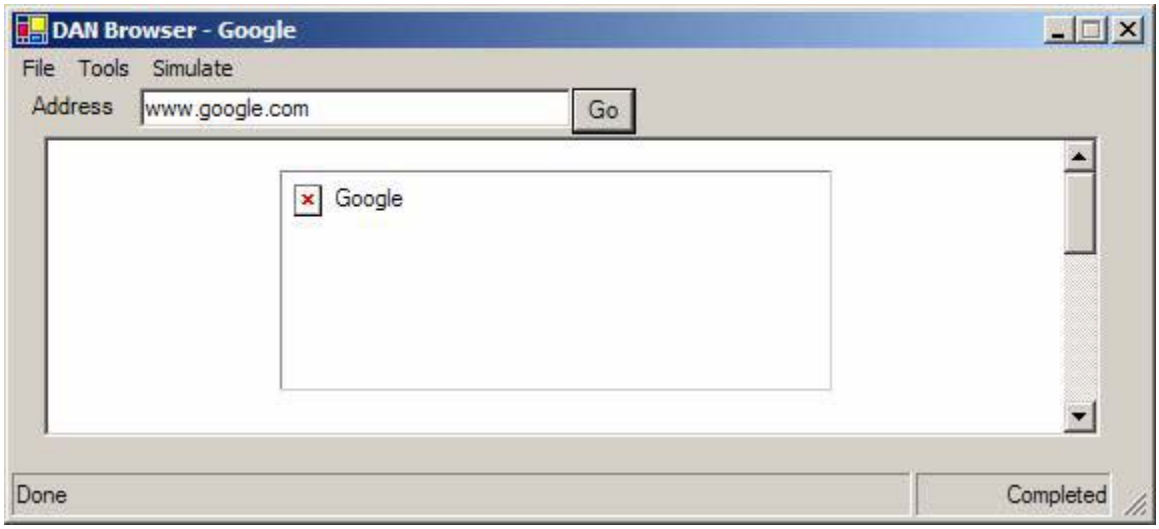

Figure 16. Cell Phone in DAN Mode –No Image

d. Notebook in DAN Mode with High/ Moderate Battery Level

The DAN browser is now set as a notebook with high or moderate level of battery. It is used to browse to Answers.com website.

The webpage is displayed in its original state. In this instance, the DPU did not attempt to compress the content since the end device is capable of displaying the content returned by the web server.

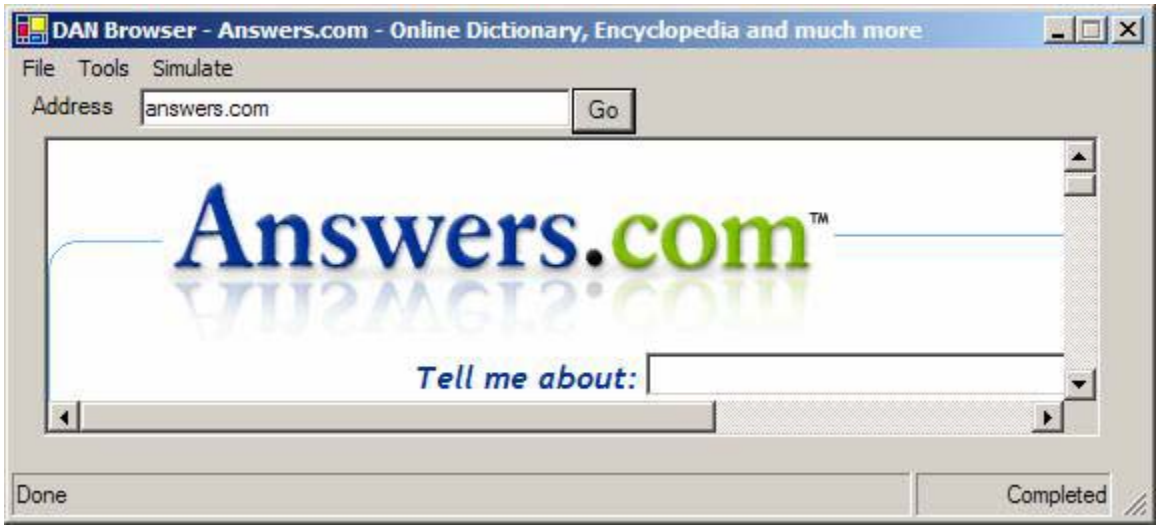

Figure 17. High/Moderate Battery Level – Full Image

e. Notebook in DAN Mode with Low Battery Level

The browser is now simulated to be running low on battery. When used to display the Answers.com website again, it is noted that the images displayed are of lower resolution.

This is because the DPU detects the low battery level of the client, and tries to conserve power consumption of the client device by sending lower resolution images.

Figure 18. Low Battery Level – Reduced Resolution Image

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

A. CHAPTER OVERVIEW

This chapter concludes our thesis and provides recommendations for further research in related areas. It covers the lessons learnt and the issues that arose during the course of our study and implementation of the prototype, followed by recommendations for further research.

B. LESSONS LEARNT

The software development environments for handheld devices are at an early stage of development. They lack the standardization and facilities needed for rapid development of programs, especially if the programs have to deal with the system level information. We faced significant challenges in developing the system and learnt a few important lessons.

1. Setting Proxy for Pocket Internet Explorer

Current Pocket PC-based PDAs use Pocket Internet Explorer (IE) for web browsing. This browser, unlike the mainstream Internet Explorer, does not provide a suitable programming interface like the WININET API. Therefore, the only way to programmatically configure the browser's proxy settings is to modify its registry entries directly. However, this method is inherently risky, and should only be performed after extensive study and testing to ascertain the appropriate entries to modify.

We carried out such testing on two iPAQ Pocket PC-based PDAs (H5550 and H4155), and successfully implemented the suggested method to programmatically configure the Pocket IE's proxy settings.

The first step involves identifying the relevant registry entries. Unlike the desktop Windows OS, the Pocket PC version does not provide any registry management tools. We used a freeware download from the internet, called the PHM Registry Tool, for this purpose. After some tedious tracking and testing, we finally managed to identify

those entries we needed to configure the proxy settings. These entries are tabulated below.

Key ***"Field = Value"***	**Remarks**
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ ConnMgr\Destinations\The Internet *"DestId = {436EF144-B4FB-4863-A041-F905A62C572}"*	This is a unique ID used to denote a endpoint of a connection. The term "destination" is a misnomer since the endpoint may refer to either the source or destination. The configuration may be system-defined (e.g., Internet, My Work Network, My ISP) or user-defined (e.g., HomeNet – see last destination entry). For system-defined destination, the ID is universal– same across all Pocket PC. This ID is used to represent destinations in field which require such information.
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ ConnMgr\Destinations\My Work Network *"DestId = {18AD9FBD-F716-ACB6-FD8A-1965DB95B814}"*	
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ ConnMgr\Destinations\My ISP *"DestId = {ADB0B001-10B5-3F39-27C6-9742E785FCD4}"*	
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ ConnMgr\Destinations\HomeNet *"DestId = {D68567FF-CF68-2F9D-C019-F9B9B9A5B554}"*	
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ ConnMgr\Providers\{EF097F4C-DC4B-4c98-8FF6-AEF805DC0E8E}\HTTP-{D68567FF-CF68-2F9D-C019-F9B9B9A5B554} *"DestId = {436EF144-B4FB-4863-A041-8F905A62C572}"* *"Enable = dword:00000001"* *"Proxy = homeproxy:80"* *"SrcId = {D68567FF-CF68-2F9D-C019-F9B9B9A5B554}"* *"Type = dword:00000001"*	The "Provider" key consists of entries used by the Pocket PC to determine if proxy should be used when connecting to the internet. There are three types of "Provider" key – HTTP, SOCKS and NULL-CORP. HTTP refers to the settings used for web connection. This entry only exists if the PDA is configured to access internet. To configure the PDA to use proxy, the proxy address should be entered in the "Proxy" field. SOCKS refers to settings for socket-based communications. Similar to the HTTP entry, it is only available if the PDA is configured to access internet. NULL-CORP is the standard setting for "My Work" endpoint..
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ConnMgr\Providers\{EF097F4C-DC4B-4c98-8FF6-AEF805DC0E8E}\SOCKS-{D68567FF-CF68-2F9D-C019-F9B9B9A5B554} *"DestId = {436EF144-B4FB-4863-A041-8F905A62C572}"* *"Enable = dword:00000001"* *"Proxy = homeproxy:80"* *"SrcId = {D68567FF-CF68-2F9D-C019-F9B9B9A5B554}"* *"Type = dword:00000004"*	

Table 21. Registry Entry for Pocket IE (Select)

Next, we needed to write an application to access and modify these registry entries. We used the Registry and RegistryKey class libraries from OpenNetCF. These classes are very easy to use since they are similar to the desktop version available in the .Net Framework. The other advantage of using these classes is that they are written in C#, thus allowing us to write the entire GUI application in C#. The code excerpts for setting the proxy is given below.

```
private void setProxy(string destID, string srcID, int protocolType,
                        string fullProxyName)
{
      RegistryKey regKey;
const string subKeyNamePrefix =
      "SOFTWARE\\Microsoft\\ConnMgr\\Providers\\{EF097F4C-DC4B-
      4c98-8FF6-AEF805DC0E8E}\\";
string subKeyName;

//Determine the appropriate SubKey Name to use.
//E.g. Modem uses HTTP or null-corp
//HTTP uses "HTTP" prefix
//Sockets uses "SOCKS" prefix
switch(protocolType)
{
      case PROTOCOL_HTTP:
            subKeyName = subKeyNamePrefix + "HTTP-" + srcID;
            break;
      case PROTOCOL_SOCKS:
            subKeyName = subKeyNamePrefix + "SOCKS-" + srcID;
            break;
      case PROTOCOL_MODEM: //currently no action required for this
type.
      default:
            return;
}

//Change the proxy settings in the specified Sub-Key
regKey = Registry.LocalMachine.CreateSubKey(subKeyName);
if (regKey != null) //Ensure a valid registry key is returned
{
      //Check if subkey already exists or newly created by
CreateSubKey().
      if (regKey.ValueCount == 0)
      {
            //Need to create these values for newly created subkey.
            //Did not include those values that are not used.
            regKey.SetValue(VALUENAME_SRCID, srcID);
            regKey.SetValue(VALUENAME_DESTID, destID);
            regKey.SetValue(VALUENAME_ENABLE, (int)1);
            regKey.SetValue(VALUENAME_TYPE, (int)protocolType);
      }
```

```
      //Update Proxy-Specific information (regardless of existing or
newly created key)
      regKey.SetValue(VALUENAME_PROXY, fullProxyName);

      regKey.Close(); //To flush the changes to the registry & close
handle in Win32 API
      // regKey.Dispose();  //Is this needed?  Not used in sample code.
}
}
```

Finally, we tested the application and found that it is able to configure the proxy settings as required.

2. Dynamic Downloading and Execution of Codes

A key strength of Java is its ability to move the program code from one computing devices to another. Once the code has been downloaded, it can be executed without prior knowledge of the methods and attributes that a class contains. The key enabler for this is the Reflection feature in Java.

The Java Reflection Application Program Interface (API) provides a means to represents objects that are components of a class file and a mechanism to obtain information about these components in a "safe and secure way" [11]. Therefore, within the bounds of the security policy, we can make use of the Reflection API to create new instances of a class, retrieve and change attributes of a class and most importantly, query and invoke public methods exposed by a class.

In our implementation of the DPU, the `JarClassLoader` class uses the Reflection API to introspect and extract the main class from the list of class files contain in a JAR file. Once the main class is identified, it makes use of the `method.invoke()` function to execute the application.

C. KNOWN ISSUES

1. Configuring DAN Browser for Session-based DPUs

Currently, setting the DPU for a particular DAN browser affects every other instances of DAN browser running on the host terminal. The effect is that other instances

of the DAN browser running on the same host terminal will all be configured to the same DPU. This is not desirable as different web sites are supposed to have different designated DPUs.

The cause of the problem is the sharing of the common registry entries by all the instances of DAN browser running in the same host. To overcome this problem, we need to establish session-based DPU settings, so that the settings for one browser will not affect others.

2. Retrieving DPU Information for Web Links and Redirected Web Sites

The current design and implementation of the browser is such that the DPU is looked up each time the user clicks the "GO" button. However, there are other ways by which a user is able to navigate to other web sites. For example, the user may click on the hyperlink on the displayed web page, or the entered URL may re-direct to another website (e.g., webpage moved to another website). The current implementation has not catered for such scenarios. As a result, websites navigated in indirect ways will not be updated with the designated DPU.

This problem may be overcome by intercepting the event when the user clicks on the hyperlink, or when the webpage is redirected to another website, just like what the browser currently does when the user clicks the "GO" button.

D. RECOMMENDATIONS

Due to time constraints, a number of areas were not examined in the implementation of the prototype and are discussed in the following paragraphs.

1. Router-based DAN Protocol

The DAN protocol devised in the prototype is not sufficient to support the Router-based DAN approach. As mentioned in Chapter 3, the router-based approach does not require a client to know the location of the DPU that is handling the DAN processing and the client connects directly to the intended information server. The router-based DAN

protocol will involve modifications of the contents of network packets. One possible implementation of this protocol involves one of the routers along the route between a client and an information server to identify itself as the DPU that will perform DAN related processing. It intercepts packets designated for the client and performs the required processing of the content before forwarding them to the client. Identification of the DPU along the route is challenging as it involves modification of the existing routing protocols.

2. DAN as a Web Service

The elements that make up the prototype include a client, a Lookup Server and the DPU. Architecturally, this setup is very similar to that of a web service [12]. We can extend the work in this thesis to make it compliant to the requirements of a web service. Moreover, making DAN a web service may make it easier to interoperate with existing and future systems.

3. Security in DAN Information Exchange

The information exchanges implemented are based on ASCII text. They are prone to eavesdropping by an authorized observer. Some form of encryption may be required to protect the confidentially of the messages. We can extend the study to use Public Key Infrastructure (PKI) for secured exchanges. Moreover, we can also harness the features of PKI to digitally sign the messages and downloaded codes to ensure their integrity and authenticity.

E. SUMMARY

The heterogeneous computing environment has resulted in the current situation whereby end-devices with differing compatibilities request for information from the same source. Without taking into consideration and matching the capabilities of the end-devices, the network may deliver content that cannot be processed by the devices. This has resulted in the end-devices and networks spending unnecessary resources to deliver and process unusable information. With that as the background, this project aimed to

develop architectures for a device-aware network that can match the capability of the end-devices to the information delivered, thereby optimizing the network resource usage.

Two possible DAN architectures, namely the Proxy-based approach and the Router-based approach, were proposed and studied in detail. In the Proxy-based approach, an immediate entity known as a DPU, is introduced and is located in between the route used for an end-device to communicate with an information server. It handles requests to an information server on behalf of the end-device and processes responses from the information server to match the end-device capability before forwarding them to the end-devices. The other approach, the Router-based approach, uses the gateway router of an information server to be the DPU. The gateway router intercepts packets designated for the client and performs the required processing of the contents before forwarding them to the client. The main objective is to prevent unusable information from entering the wide area network.

In order to demonstrate the feasibility of the architecture, a prototype has been developed. We have successfully demonstrated how the DAN framework can enhance the network by enabling it to differentiate the client devices so as to deliver the optimal content to them. In addition, it is able to respond appropriately to the device's dynamic profile as well.

In conclusion, the objectives of this project were met. We hope that the results and prototype developed from this project can provide a valuable reference for further development of the DAN framework.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. "Jini Technology Architecture Overview", http://www.sun.com/software/jini/whitepapers/architecture.html, last accessed on February 2005.

2. Jan Newmarch, *A Programmer's Guide to Jini Technology*, APress, Berkeley, California, 2000.

3. Gurminder Singh, *Content Repurposing*, IEEE Multimedia, vol. 11, no. 1, pp.20-21, January-March 2004.

5. "Resource Description Framework", http://www.w3.org/RDF/, last accessed on December 2004

6. "User Agent Profile Specification", Wireless Application Group User Agent Profile Specification, Version 10-Nov-1999

7. "RFC 2131 – Dynamic Host Configuration Protocol", http://www.ietf.org/rfc/rfc2131.txt, last accessed on March 2005

8. "RabbIT Web Proxy", http://rabbit-proxy.sourceforge.net, last accessed on March 2005

9. "Using JAR-related APIs from the Java Tutorial", http://java.sun.com/docs/books/tutorial/jar/api/jarclassloader.html, last accessed on February 2005

10. "ImageMagick", http://www.imagemagick.org, last accessed on February 2005

11. "Take an in-depth look at the Java Reflection API", http://www.javaworld.com/javaworld/jw-09-1997/jw-09-indepth.html, last accessed on March 2005

12. "W3C Working Group Note - Web Services Architecture", http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, last accessed on March 2005

13. Su Wen, Gurminder Singh, John Gibson and Arijit Das. Towards Device-Aware Networks. The 12th Internation Conference on Telecommunications Systems (ICTMS12). Monterey, California, July 2004.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Gurminder Singh
   Naval Postgraduate School
   Monterey, California

4. Assistant Professor Su Wen
   Naval Postgraduate School
   Monterey, California